

# **EXHIBIT 1**

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

THE HONORABLE MARSHA J. PECHMAN

IN THE UNITED STATES DISTRICT COURT  
FOR THE WESTERN DISTRICT OF WASHINGTON  
SEATTLE DIVISION

REC SOFTWARE USA, INC., a Virginia  
corporation,

Plaintiff/Counter-Defendant,

v.

DELL INC., a Delaware corporation; and  
DELL PRODUCTS L.P., a Texas limited  
partnership,

Defendants/Counter-Plaintiffs.

Case No. 2:14-cv-01048-MJP

**REC SOFTWARE USA, INC.'S  
DISCLOSURE OF ASSERTED  
CLAIMS AND INFRINGEMENT  
CONTENTIONS**

**REC SOFTWARE USA, INC.'S DISCLOSURE OF  
ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS**

Pursuant to Local Patent Rule 120 and the Court's October 9, 2014, Scheduling Conference, plaintiff REC Software USA, Inc. ("REC") hereby submits its Disclosure of Asserted Claims and Preliminary Infringement Contentions. This disclosure and the assertions and contentions set forth herein are based on the information available to REC as of the date of hereof, and REC reserves the right to amend this disclosure and the assertions and contentions set forth herein to the fullest extent consistent with the Court's Rules and Orders. This disclosure and the assertions and contentions set forth herein are designated and marked "CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY" with the understanding that counsel may disclose the contentions to one in-house counsel of Defendant if such one in-house counsel of Defendant agrees to maintain in confidence this disclosure and the assertions and

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

1 contentions set forth herein. *See, e.g.*, Memorandum Opinion and Order [Docket No. 330],  
 2 *ExitExchange Corp. v. Casale Media Inc., et al.*, United States District Court for the Eastern  
 3 District of Texas, Case No. 2:10-cv-00297-JRG (March 23, 2012).

4 **1. Right to Supplement**

5 REC bases this disclosure and the assertions and contentions set forth herein on its  
 6 current knowledge, understanding, and belief as to the facts and information available to it as of  
 7 the date hereof. This case is still in the early stages of discovery and REC has not yet completed  
 8 its investigation, collection of information, discovery, or analysis related to this action.

9 Accordingly, REC reserves the right to supplement, amend, and/or modify the information  
 10 contained herein and to use and introduce such information and any subsequently-identified  
 11 documents in this case, including at trial. In particular, REC reserves its right to supplement,  
 12 amend, and/or modify its identification and disclosure of asserted claims and its identification  
 13 and disclosure of accused products. Additionally, as further discovery is taken, and additional  
 14 details are provided regarding Defendant's activities, REC's infringement charts and contentions  
 15 may need to be amended, supplemented, modified, and/or corrected. REC also reserves its right  
 16 to supplement its disclosure of documents based upon further investigation and discovery.

17 **2. Asserted Claims**

18 Based upon its current knowledge, understanding, and belief as to the facts and  
 19 information available to it as of the date hereof, REC asserts claims 1, 2, 3, 4 and 6 of U.S.  
 20 Patent No. 5,854,936 (the "Asserted Claims") against defendants DELL INC. and DELL  
 21 PRODUCTS L.P (referred to in this disclosure separately and collectively as "Defendant").  
 22 REC reserves the right to assert additional claims against Defendant based upon results of  
 23 discovery and further investigation. REC does not reserve the right to assert any of claims 9-22  
 24 of U.S. Patent No. 5,854,936 against Defendant.

25 The Asserted Claims involve features that are implemented by software and REC's  
 26 current positions on infringement are set forth without the benefit of acquiring the code for the

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

1 Accused Instrumentalities. Therefore, it may be necessary for REC to supplement its position on  
 2 infringement after a complete production of code by Defendant.

3 REC's position with respect to the infringement of specific claims will depend on the  
 4 claim construction adopted by the Court. REC's position with respect to the infringement of the  
 5 Asserted Claims is based on the claim construction set forth in the Order on Claim Construction  
 6 [Doc. No. 159] entered in *REC Software USA, INC. v. Bamboo Solutions Corporation, et al.*,  
 7 Case No. 2:11-cv-00554-JLR (U.S. District Court for the Western District of Washington), and  
 8 the agreed terms set forth in the Supplemental Joint Claim Construction and Prehearing  
 9 Statement [Doc. No. 134] filed in the same case. Because the Court may modify such claim  
 10 construction and agreed terms or otherwise not adopt such claim construction and agreed terms  
 11 in their entirety, REC cannot take a final position on the bases for infringement of the Asserted  
 12 Claims.

13 Plaintiff's investigation and analysis of Defendant's products and services is based upon  
 14 on publicly available information. Plaintiff reserves the right to seek leave of court to amend this  
 15 disclosure and the assertions and contentions set forth herein based upon discovery of non-public  
 16 information.

17 Plaintiff reserves the right to seek leave of court to amend these contentions based upon  
 18 non-infringement positions taken by Defendant, if any, together with invalidity positions taken  
 19 by Defendant, if any.

20 **3. Accused Instrumentalities & Comparison To Asserted Claims**

21 The chart attached hereto as Appendix A identifies the accused instrumentalities, which  
 22 relate to Defendant's mobile phones and/or tablets and which may be referred to in this  
 23 disclosure as the "Accused Devices" or the "Accused Systems", and in combination compares  
 24 them, element-by-element, to the above-listed Asserted Claims.

25 Defendant is liable for direct infringement of the asserted claims. Defendant's Accused  
 26 Devices include software that dictate the performance of such Accused Devices, including

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

1 without limitation, automatically performing the steps of the Asserted Claims. Some non-  
2 exclusive examples of Defendant's direct infringement include: Making the Accused Devices;  
3 Testing the Accused Devices; Using the Accused Devices; Offering to sell the Accused Devices;  
4 Selling the Accused Devices; Importing the Accused Devices; and Developing and testing multi-  
5 module programs for the Accused Devices.

6 To the extent that any element of any Asserted Claim is performed, or alleged by  
7 Defendant to be performed, by a party or entity other than Defendant (*e.g.*, joint infringement),  
8 REC asserts that such other party or entity performs such element under the direction and/or  
9 control of Defendant.

10 With respect to joint infringement based upon joint acts of multiple parties or entities,  
11 Defendant directs its customers (*e.g.*, other party or entity) to perform the rest.

12 With respect to joint infringement based upon joint acts of multiple parties or entities,  
13 Defendant directs and/or controls the other party (*e.g.*, other party or entity).

14 With respect to joint infringement based upon joint acts of multiple parties or entities,  
15 Defendant performed and/or is responsible for the performance of the claims.

16 With respect to joint infringement based upon joint acts of multiple parties or entities,  
17 Defendant provides instructions and/or directions to another party or entity for performing the  
18 steps of the claims.

19 With respect to joint infringement based upon joint acts of multiple parties or entities  
20 based upon cooperative acts of the other party or entity.

21 With respect to joint infringement based upon joint acts of multiple parties or entity,  
22 Defendant directs and/or knowingly works with another to bring about the performance of the  
23 claims.

24 With respect to direct infringement based on joint acts of multiple parties, the role of each  
25 such party in the direct infringement is described in the chart attached hereto.

26 Defendant may also be engaged in infringing activities in connection with products

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

1 and/or services of which REC is not aware. REC reserves the right to amend its disclosure and  
2 the assertions and contentions set forth herein to include any other infringing activities that are  
3 identified through further investigation and discovery.

4 **4. Literal and Equivalents Infringement**

5 As supported and explained in the attached chart, it is currently believed that each of the  
6 elements of the Asserted Claims of the '936 patent is met literally, and if any claim limitation is  
7 not met literally, then it is met under the doctrine of equivalents. It is expected that the same  
8 facts upon which REC's literal infringement claim is based will also form the basis of REC's  
9 doctrine of equivalents claim, as any differences between the limitations of the Asserted Claims  
10 and the Accused Devices are insubstantial. With respect to the doctrine of equivalents, however,  
11 as Defendant has not yet provided details of its non-infringement positions, REC reserves the  
12 right to present further facts to support assertions of infringement under the doctrine of  
13 equivalents.

14 **5. Priority Date**

15 The Asserted Claims of the '936 patent claim priority to U.S. Patent No. 5,649,204, filed  
16 on August 22, 1991.

17 DATED this 7th day of November, 2014.

18 STOLL STOLL BERNE LOKTING & SHLACHTER P.C.

19 By: s/Timothy S. DeJong

20 **Timothy S. DeJong**, WSBA No. 20941  
21 209 SW Oak Street, Suite 500  
22 Portland, OR 97204  
Telephone: (503) 227-1600  
Facsimile: (503) 227-6840  
Email: tdejong@stollberne.com

23 **Attorneys for Plaintiff REC Software USA, Inc.**

**CERTIFICATE OF SERVICE**

I hereby certify that I caused to be served the foregoing **REC SOFTWARE USA, INC.'S DISCLOSURE OF ASSERTED CLAIMS AND INFRINGEMENT**

**CONTENTIONS** on the following named person(s) on the date indicated below,

☒ By U.S. Mail with postage prepaid

to said persons a true copy thereof, and if by mail, contained in a sealed envelope, addressed to said person at their last known addresses indicated below.

Michael R. Scott  
Eric D. Lansverk  
1221 Second Avenue, Suite 500  
Seattle, Washington 98101-2925  
Email: michael.scott@hcmp.com;  
eric.lansverk@hcmp.com

**Attorneys for Defendants**

DATED this 7th day of November, 2014.

**STOLL STOLL BERNE LOKTING & SHLACHTER P.C.**

By: s/Timothy S. DeJong

**Timothy S. DeJong**, WSBA No. 20941  
209 SW Oak Street, Suite 500  
Portland, OR 97204  
Telephone: (503) 227-1600  
Email: tdejong@stollberne.com

**Attorneys for Plaintiff/Counter-Defendant  
REC Software USA, Inc.**

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

IN THE UNITED STATES DISTRICT COURT  
FOR THE WESTERN DISTRICT OF WASHINGTON  
SEATTLE DIVISION

**Appendix A**

The following analysis is based on information known to date, as well as the Order on Claim Construction [Doc. No. 159] entered in *REC Software USA, INC. v. Bamboo Solutions Corporation, et al.*, Case No. 2:11-cv-00554-JLR (U.S. District Court for the Western District of Washington) and the Supplemental Joint Claim Construction and Prehearing Statement [Doc. No. 134] filed in the same case. Plaintiff reserves the right to amend and update this analysis as additional information is obtained and analyzed.

The products analyzed (referred to herein collectively as the “Accused Devices” or the “Accused Systems”) are Defendant’s mobile phones and/or tablets that include and implement at least (i) an operating system, (ii) certain pre-installed application programs, and (iii) a Dalvik Virtual Machine (“Dalvik VM”), including any Dalvik VM included with any one or more of the following versions of the Android Operating System software (the “Android system”):

Android 1.0		(API level 1)
Android 1.1		(API level 2)
Android 1.5	(Cupcake)	(API level 3)
Android 1.6	(Donut)	(API level 4)
Android 2.0	(Éclair)	(API level 5)
Android 2.0.1	(Éclair)	(API level 6)
Android 2.1	(Éclair)	(API level 7)
Android 2.2-2.2.3	(Froyo)	(API level 8)
Android 2.3-2.3.2	(Gingerbread)	(API level 9)
Android 2.3.3-2.3.7	(Gingerbread)	(API level 10)
Android 3.0	(Honeycomb)	(API level 11)
Android 3.1	(Honeycomb)	(API level 12)
Android 3.2-3.2.4	(Honeycomb)	(API level 13)
Android 4.0-4.0.2	(Ice Cream Sandwich)	(API level 14)
Android 4.0.3	(Ice Cream Sandwich)	(API level 15)

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The default Android Operating System, including the Dalvik VM, for version of the Android system is provided in Appendix C produced herewith.

In particular, the Accused Systems include the mobile phones and/or tablets identified in Appendix B, produced herewith, that include and implement any one or more of the pre-installed application programs identified in Appendix B, and any one or more of the operating systems identified in Appendix B; provided, however, that:

- the listed phones and/or tablets are intended to be merely exemplary, and other phones and/or tablets are intended to be included within the scope of these infringement contentions to the extent applicable;
- variations of the listed phones and/or tablets, including those having hardware and/or software different than those listed, are intended to be included within the scope of these infringement contentions to the extent applicable;
- the listed pre-installed application programs are intended to be merely exemplary, and other pre-installed application programs are intended to be included within the scope of these infringement contentions to the extent applicable;
- variations of the listed pre-installed application programs are intended to be included within the scope of these infringement contentions to the extent applicable; and
- variations of the listed operating system, including the Dalvik VM, are intended to be within the scope of these infringement contentions to the extent applicable.

**Background on Technology**

This section provides some basic background information concerning computers (*e.g.*, a computing device, such as mobile phones and/or tablet) and how computer programs are executed on computers. This background section provides context to better understand the

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

inventions claimed by U.S. Patent No. 5,854,936 (referred to herein as the “ ‘936 patent”), entitled “Code server”.

**What is a Mobile Phone and/or Tablet?**

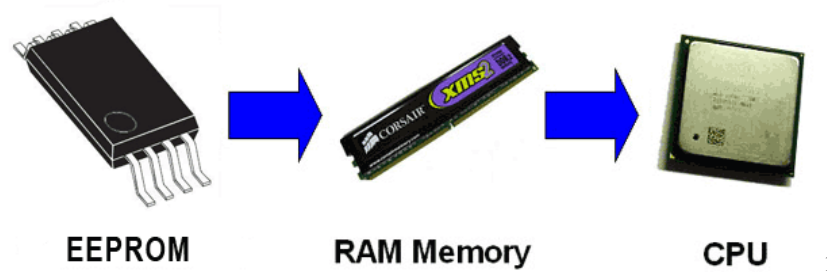
A mobile phone and/or tablet is a computing device, generally referred to as a computer, that follows a list of instructions to perform some useful operation or operations. Those instructions are called a program and are generally referred to as a computer program. The actual computing device consists of pieces of electronic gadgetry made out of silicon housed in an enclosure. All these parts of the mobile phone and/or tablet that we can see and touch we call hardware. The program is not made of silicon or aluminum or steel. Its instructions are encoded as magnetic or electric charges. Computer programs are of course known as software. The charges in software represent two things: positive or negative; on or off; 0 or 1. A 0 or 1 is known as a bit. Eight bits put together is called a byte. Representations of bytes may look something like this: 00101110 10100010 11001001 and so on. A byte can represent a number or an alphanumeric character or a logical operation. Computer instructions like these are often referred to as machine code, but computer instructions can also take the form of virtual code, representing the same instructions in a form that is easily adaptable to computers which understand differing machine codes.

**What is hardware?**

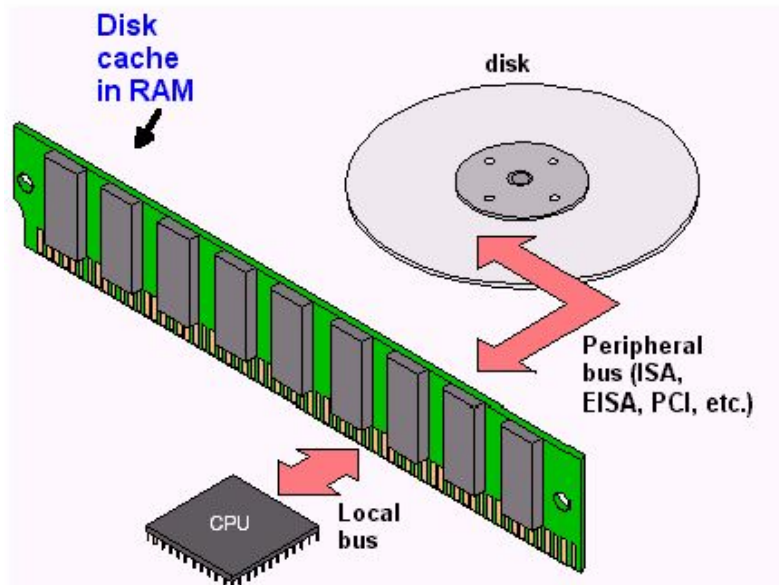
There are three main pieces of hardware among the components of a mobile phone and/or tablet, used for the running of programs. These are a CPU (central processing unit), a RAM (volatile memory), and a non-volatile storage device such as EEPROM (electrically erasable programmable read only memory) or a disk drive (*see, e.g.*, the figures below). Computing devices frequently have other components, including components for input, such as keyboard

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

and touch screens, and components for output, such as displays and speakers. Only a CPU, RAM, and a storage device are necessary to run a program.



From Computer Desktop Encyclopedia  
© 1999 The Computer Language Co. Inc.



Although the storage device itself is hardware, its implementation and organization is implemented by software. For example, the Android system in the Accused Products stores information on the storage device, but the Android system itself, including the Dalvik VM, is implemented as software. That is, the architecture and organizational structure of the Android system is coded by software.

<sup>1</sup> <http://www.hardwaresecrets.com/article/209>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

**Running programs.**

A mobile phone and/or tablet runs a program as follows: The CPU fetches an instruction from RAM or a non-volatile storage device and performs (“executes”) the instruction. The instruction might tell the CPU to add two numbers or to output a character to the screen or to read a key. But the main point is that while a CPU is executing a program, the instructions are generally fetched from RAM or a non-volatile storage device.

Prior to execution, instructions must be loaded or “mapped” from the RAM or non-volatile storage device so that the CPU can execute the instructions (run the program). “Mapping” makes instructions in non-volatile storage available to the CPU for execution by assigning addresses to those instructions that correspond to RAM areas in process memory address space.

As a practical matter, during the execution of most modern software programs the CPU is constantly loading instructions, executing the instructions, then fetching additional instructions. So the speed at which a computer can load instructions is critical to efficient operation, and, likewise, existing knowledge of the code modules may be used to increase efficient operations. It is just such a technique that the ‘936 patent is directed to, namely, the efficient preparing of a program for execution.

**Different types of software.**

Some software is designed to perform a specific task. For example, a clock program’s main purpose is indicating the current time. You surf the web using a browser program. You check your e-mail using an e-mail program. These are examples of application software – programs or collections of programs that are designed to accomplish certain purposes.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Other software is required to help the CPU load instructions from the storage device, communicate with peripherals like screens and speakers, and manage RAM usage. The Android Operating System together with portions of the Dalvik VM is this type of software. An operating system provides the environment within which application programs run.

Consider the following example of how an application works together with the operating system of the computer: You open up an email application and type an email and then send it.

Several components work together to make this happen: The keyboard and/or touch screen of the phone and/or tablet send your input to the operating system. The operating system determines that the email application is the active application and accepts your input as data for that application. Each instruction from the programs that make up the email application is sent by the operating system to the CPU. These instructions may be intertwined with instructions from other programs that the operating system is overseeing before being sent to the CPU. All this time, the operating system is steadily providing display information directing what will be displayed on the screen. When you choose to send the email, a request is sent to the operating system, which then sends the email. Once you have sent the email, the operating system also directs the data to the appropriate storage device to save the email.

Consider another example of how an application works together with the operating system on the computer: You turn on your phone and/or tablet and the clock automatically loads and executes. Several components work together to make this happen: Turning on the power to your phone and/or tablet results in starting its operating system. When the operating system has completed loading then the clock program is automatically triggered and the clock program is executed. The clock program then displays the current time and date on the display of the phone and/or tablet.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****Multi-Module Programs.**

As programs became more complex and larger, programmers began writing programs in piecemeal fashion. That is, the programs were written as smaller units of computer instructions (known as modules) that link to other like units of computer instructions (other modules) to form multi-module programs.

For example, a program may consist of module A which in turns links to module B. When this program is executed, module A is first executed and then module B is subsequently executed. This results in the same instructions being performed (*i.e.*, the instructions of module A and module B) that would have been performed if the program was not written in multi-module form but was instead written as a single module program that included all the instructions of modules A and B.

Multi-module programs can also be more complex and do not always run in a linear fashion. Consider the exemplary program described by the '936 patent that consists of several modules: a main code module, module A, module B, module C, etc.<sup>2</sup> The program, when executed, does not simply run the main code module, followed by module A, followed by module B, followed by module C, etc. To the contrary, a module in a multi-module program typically includes information indicating the modules to which it links (*i.e.*, references to other code modules). In the same example, the main code module includes references to, and links to, three other code modules – modules A, B, and C. These modules in turn link to other code modules. Module B includes a reference to, and links to, module E. Module C includes a reference to, and links to, module B.

---

<sup>2</sup> See '936 patent, Figure 2 and accompanying text.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Executing programs with modules that include references to any of several other modules allows for more complex programs to be written and executed without the need for internal redundancy. That is, instead of including the complete code for module B each time the functionality of module B is called for, the code can simply include a reference to module B.

Given the complexity of modern programs and applications, the ability to cross reference other modules is a necessary functionality to prevent programs from becoming exceedingly long and unmanageable, and unnecessarily occupying more memory. A particular module that references another module or modules is sometimes unable to run, or be executed, in its entirety without the modules that it references, or links to, also being executed. For this reason, the referenced modules are often referred to as dependencies.

For example, in an accounting application, a multi-module program may calculate an individual's taxes for a particular year. One particular module – call it the “main code module” may include the programming functionality of determining the total amount of deductions. This main code module may include references to several other code modules, including, for example, a code module that calculates the amount of home mortgage that can be deducted and a code module that calculates the amount of charitable contributions that can be deducted. If the module that calculates home mortgage could not be located and executed, the main code module would not run properly. It would either fail to include the home mortgage deduction or be unable to execute at all and render an error message.

**The ‘936 Patent**

The ‘936 patent teaches a novel way of maintaining and preparing multi-module programs for execution. A benefit to the claimed invention is a more efficient way of preparing multi-module programs for execution. The ‘936 patent teaches a more efficient way to gather

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

information concerning which other modules each module links to, a necessary step when running a multi-module program. In particular, the patent teaches a “code server [that] includes a code module information table [] which stores information that links modules of coded programs together.”<sup>3</sup> Accordingly, to gather information identifying which modules of the multi-module program a particular module links to, a system implementing this teaching of the ‘936 patent need not extract this information from the code module itself. Instead of the inefficient extraction process, this information can be looked up in the module information table.

Defendant’s Accused Systems have implemented this teaching of the ‘936 patent by avoiding the inefficient process of scanning and extracting linkage information from within the executable code and, instead, determining the linkage information from a resource pointer table(s).

**Operation of the Accused Systems**

In this section, various sources of information are cited about the operation of the Accused Systems. Rather than repeat this section throughout the infringement contentions, this section is incorporated by reference into the analysis of each element (as appropriate) and sub-element (as appropriate) of the asserted claims.

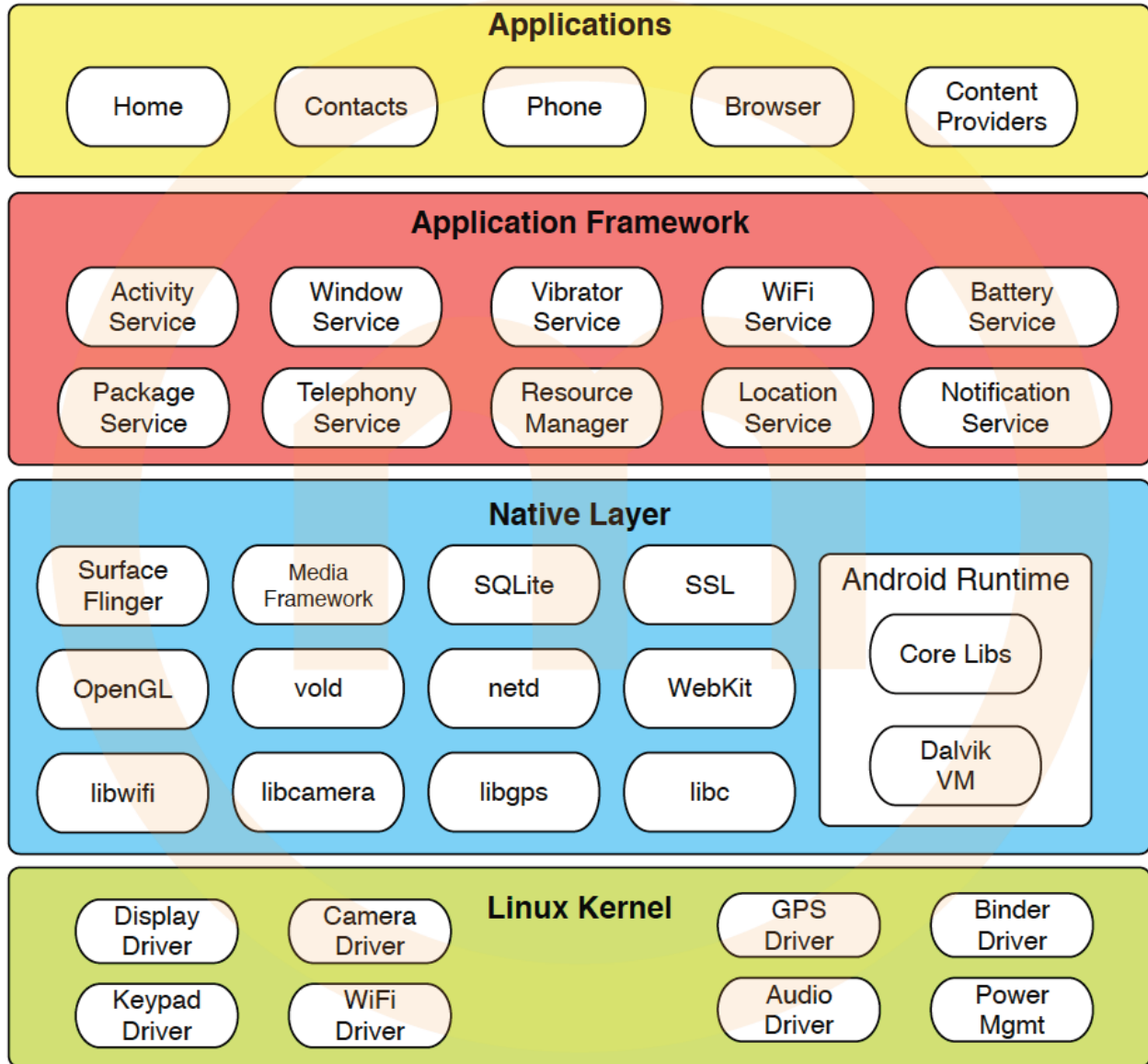
**The Android Software “Stack”**

In the architecture of Android system, one layer builds upon another. The software layers comprise the “Android Stack” depicted in the figure below.<sup>4</sup>

---

<sup>3</sup> ‘936 patent, 3:49-51.

<sup>4</sup> The Android Stack [from “Android Services Black Magic”, p. 3, by Aleksandar (Saša) Gargenta, Marakana Inc., Android Builders Summit, February 14th, 2012 (available at [http://marakana.com/s/android\\_services\\_black\\_magic\\_from\\_android\\_builders\\_summit\\_2012,1057/index.html](http://marakana.com/s/android_services_black_magic_from_android_builders_summit_2012,1057/index.html)); *see also* <https://www.youtube.com/watch?v=8d8BtRHICtM> and <https://thenewcircle.com/s/post/1057/MarakanaAndroidServicesBlackMagic.pdf>].

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****Android Stack**

Application programs sit in the “Applications” layer at the top of the stack. Applications are executed (run) directly within the Dalvik VM and have access to the Application Framework as needed. Android applications start out as Java code. The Dalvik VM executes applications

The “Application Framework” layer contains additional code that can be called from applications in the “Application” layer. The “Native Layer” contains libraries that do not run inside the Dalvik VM, but rather are called from the underlying “Linux Kernel” layer. The

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

“Linux Kernel” layer sits at the bottom of the stack and provides an interface to the device hardware.

**The Dalvik Virtual Machine**

The Dalvik VM<sup>5</sup> is a virtual machine that runs on the Accused Devices that include and implement the Android system. The Dalvik VM is included as an integrated component of the Accused Devices that include and implement the Android system. The Dalvik VM is utilized by the operating systems of the Accused Devices that include and implement the Android system. The Accused Devices include one or more versions of the Dalvik VM that correspond to the one or more of versions of the Android system included and implemented by the Accused Devices (*see* Appendix C, produced herewith).

Versions 1.5 to 2.3.7 and 4.0 to 4.0.3 of the Android system do not differ from one another in any way material to these infringement contentions. Although different versions of the Android system may include small changes to the code language, the operations of the Android system code as relevant to these infringement contentions do not differ amongst these versions.

---

<sup>5</sup> “The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory-mappable execution. The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included ‘dx’ tool. The VM runs on top of Posix-compliant operating systems, which it relies on for underlying functionality (such as threading and low level memory management). The Dalvik core class library is intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device.”

<http://developer.android.com/guide/appendix/glossary.html>; *see also*, <http://stackoverflow.com/questions/13361293/how-does-mterp-dalvik-vm-organize-its-byte-code-interpret-loop>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

It is believed that versions 1.0 and 1.1 of the Android system do not differ from the other versions of the Android system in a way that affects the infringement analysis, but the source code is not currently publically available.

It is believed that versions 3.0, 3.1, and 3.2-3.2.4 of the Android system do not differ from the other versions of the Android system in a way that affects the infringement analysis, but the source code is not currently publically available.

**Class, Dex, and Apk Files**

An Android program (or application, sometimes referred to as an “app”) is contained in an .apk file<sup>6</sup> that includes one or more .dex files.<sup>7</sup> Each .dex file is prepared from one or more .class files (*i.e.*, compiled from Java source code), which are the fundamental building blocks of programs that run on a device. These .class files are modules that link to one another as discussed in the Background section, above. The Dalvik VM loads or “maps” and executes multiple classes from the .apk file. A .dex file that includes .class files as used with the Dalvik VM differ from traditional modules in that they have another component. In particular, in addition to executable code (*i.e.*, computer instructions like those found in a traditional module), a .dex file also includes metadata that stores information relating to the .class files and its contents. Although a .class file typically contains only one class, a .dex file typically contains multiple classes. Accordingly, the two separate components of a typical .dex file are (i) the executable modules and (ii) the metadata relating to the modules.

---

<sup>6</sup> “Each Android application is compiled and packaged in a single file that includes all of the application’s code (.dex files), resources, assets, and manifest file ... For convenience, an application package file is often referred to as an ‘.apk’.”

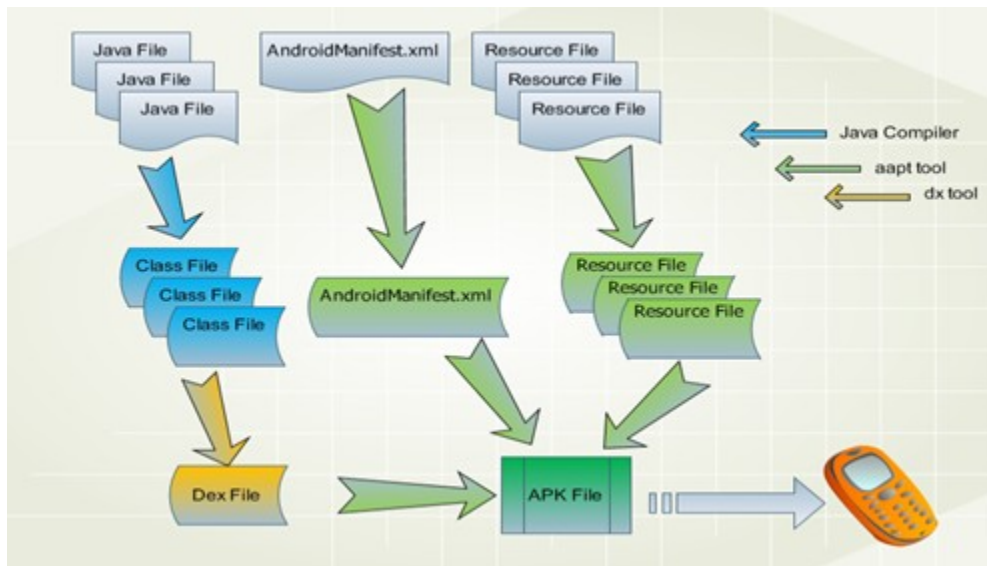
<http://developer.android.com/guide/appendix/glossary.html>.

<sup>7</sup> “Android programs are compiled into .dex (Dalvik Executable) files ... [which] can be created by automatically translating compiled applications written in the Java programming language.”

<http://developer.android.com/guide/appendix/glossary.html>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The diagram below<sup>8</sup> depicts the logical flow of the creation of an .apk file which contains the executable class files (*i.e.*, modules) and metadata therein:



The .dex file is combined with the AndroidManifest.xml<sup>9</sup> which presents information about the application to the Android system, information the system must have before it can run the application's code, and Resource File(s)<sup>10</sup>.

A more detailed look at the build process is illustrated below. The build process involves many tools and processes that generate intermediate files on the way to producing an .apk file. If you are developing in an integrated development environment ("IDE") such as Eclipse, the complete build process is automatically done periodically as you develop and save your code changes. If you are using other IDEs, this build process is done every time the generated Ant build script is run for the project. It is useful, however, to understand what is happening "under the hood" since much of the IDE build process is masked from the user.

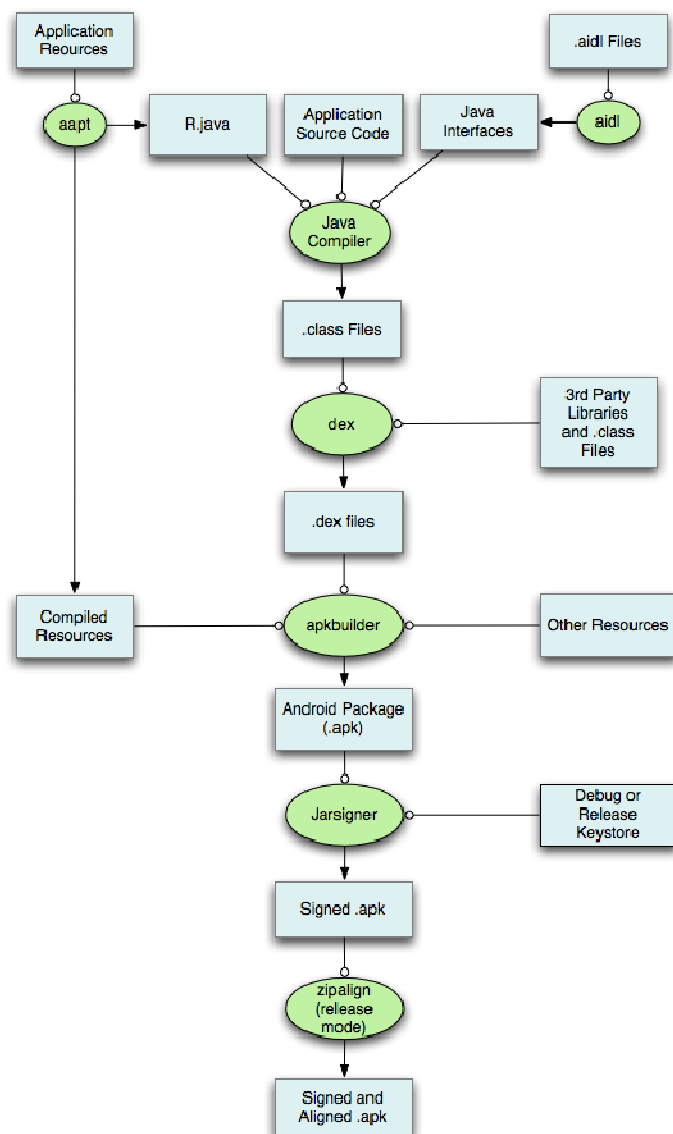
<sup>8</sup> [http://edesign.ads.ie.u-ryukyu.ac.jp/?page\\_id=369](http://edesign.ads.ie.u-ryukyu.ac.jp/?page_id=369).

<sup>9</sup> <http://developer.android.com/guide/topics/manifest/manifest-intro.html>.

<sup>10</sup> <http://developer.android.com/guide/topics/resources/index.html>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The following diagram<sup>11</sup> depicts the tools and processes involved in an .apk file “build”:



Java files are compiled by a Java compiler into corresponding .class files (in this context, “compiled” means converting .java text files into the Java “bytecode” found in .class files). The .class files, along with any third party libraries, are then compiled by a dex tool to produce one or

<sup>11</sup> <http://developer.android.com/images/build.png>;  
<http://developer.android.com/tools/building/index.html>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

more .dex files (in this context, “compiled” means converting Java bytecode to dex bytecode).<sup>12</sup>

An AndroidManifest.xml file<sup>13</sup>, together with resources files<sup>14</sup>, are combined together with the one or more .dex files by an Android Asset Packaging Tool (aapt) to create the .apk file that is then installed on the mobile phone and/or tablet.<sup>15</sup>

The following simplified diagram<sup>16</sup> depicts the components involved in building and running an Android application:

---

<sup>12</sup> The “Android Developers” website outlines “[t]he general process for a typical build” as follows:

- The Android Asset Packaging Tool (aapt) takes your application resource files, such as the AndroidManifest.xml file and the XML files for your Activities, and compiles them. An R.java is also produced so you can reference your resources from your Java code.
- The aidl tool converts any .aidl interfaces that you have into Java interfaces.
- All of your Java code, including the R.java and .aidl files, are compiled by the Java compiler and .class files are output.
- The dex tool converts the .class files to Dalvik byte code. Any 3rd party libraries and .class files that you have included in your project are also converted into .dex files so that they can be packaged into the final .apk file.
- All non-compiled resources (such as images), compiled resources, and the .dex files are sent to the apkbuilder tool to be packaged into an .apk file.
- Once the .apk is built, it must be signed with either a debug or release key before it can be installed to a device.
- Finally, if the application is being signed in release mode, you must align the .apk with the zipalign tool. Aligning the final .apk decreases memory usage when the application is running on a device.

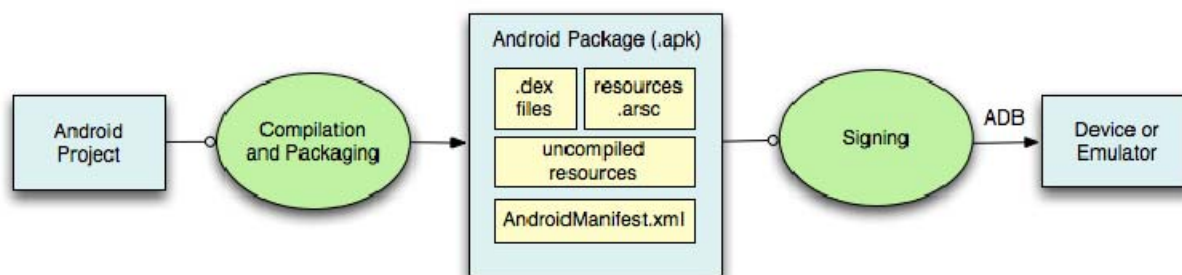
<http://developer.android.com/tools/building/index.html>.

<sup>13</sup> <http://developer.android.com/guide/topics/manifest/manifest-intro.html>.

<sup>14</sup> <http://developer.android.com/guide/topics/resources/index.html>.

<sup>15</sup> <http://developer.android.com/tools/building/index.html>.

<sup>16</sup> <http://developer.android.com/images/build-simplified.png>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

In this manner, an Android project, through the process of compilation and packing, results in an AndroidPackage (.apk file) that includes .dex files, resources .arsc, uncompiled resources, and an AndroidManifest.xml. The Android Package may be subsequently signed.

Dex files are used to hold a set of class definitions and their associated adjunct data, including metadata that describes linkages between the classes. For all classes (modules) contained in the .dex file, metadata fields string\_ids, type\_ids, proto\_ids, field\_ids, method\_ids, etc. (*see* Chart 1, “Dex File Structure”, below) contain information extracted from the classes (modules) that identify inter-related classes in the .dex file and the methods and variables defined in the classes.

Chart 1. Dex File Structure<sup>17</sup>

Name	Format	Description
header	header_item	the header
string_ids	string_id_item[]	string identifiers list. These are identifiers for all the strings used by this file, either for internal naming (e.g., type descriptors) or as constant objects referred to by code. This list must be sorted by string contents, using UTF-16 code point values (not in a locale-sensitive manner), and it must not contain any duplicate entries.
type_ids	type_id_item[]	type identifiers list. These are identifiers for all types (classes, arrays, or primitive types) referred to by this file, whether defined in the file or not. This list must be sorted by string_id index, and it must not contain any duplicate entries.
proto_ids	proto_id_item[]	method prototype identifiers list. These are identifiers for all prototypes referred to by this file. This list must be sorted in return-type (by type_id

<sup>17</sup> <http://source.android.com/devices/tech/dalvik/dex-format.html>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**Chart 1. Dex File Structure<sup>17</sup>

Name	Format	Description
		index) major order, and then by arguments (also by type_id index). The list must not contain any duplicate entries.
field_ids	field_id_item[]	field identifiers list. These are identifiers for all fields referred to by this file, whether defined in the file or not. This list must be sorted, where the defining type (by type_id index) is the major order, field name (by string_id index) is the intermediate order, and type (by type_id index) is the minor order. The list must not contain any duplicate entries.
method_ids	method_id_item[]	method identifiers list. These are identifiers for all methods referred to by this file, whether defined in the file or not. This list must be sorted, where the defining type (by type_id index) is the major order, method name (by string_id index) is the intermediate order, and method prototype (by proto_id index) is the minor order. The list must not contain any duplicate entries.
class_defs	class_def_item[]	class definitions list. The classes must be ordered such that a given class's superclass and implemented interfaces appear in the list earlier than the referring class. Furthermore, it is invalid for a definition for the same-named class to appear more than once in the list.
data	ubyte[]	data area, containing all the support data for the tables listed above. Different items have different alignment requirements, and padding bytes are inserted before each item if necessary to achieve proper alignment.
link_data	ubyte[]	data used in statically linked files. The format of the data in this section is left unspecified by this document. This section is empty in unlinked files, and runtime implementations may use it as they see fit.

The dex tool converts a developer's .class files to Dalvik byte code. Any third party libraries and .class files that the developer has included in his project are also converted into .dex files so that they can be packaged into the final .apk file. Hence, the resulting apk file may contain one or more dex files that describe relationships between the classes (modules) included in the .apk file.<sup>18</sup>

**Dex Bytecode**

The figure below illustrates a binary .dex file shown in hexadecimal format.<sup>19</sup>

<sup>18</sup> <http://developer.android.com/tools/building/index.html>.

<sup>19</sup> *Decompiling Android*, Chapter 3, by Godfrey Nolan, Apress, 2012 (showing a hexadecimal view of a classes.dex file).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

64 65 78 0A 30 33 35 00 62 8B 44 18 DA A9 21 CA
9C 4F B4 C5 21 D7 77 BC 2A 18 4A 38 0D A2 AA FE
50 04 00 00 70 00 00 00 78 56 34 12 00 00 00 00
00 00 00 00 A4 03 00 00 1A 00 00 00 70 00 00 00
0A 00 00 00 D8 00 00 00 07 00 00 00 00 01 00 00
03 00 00 00 54 01 00 00 09 00 00 00 6C 01 00 00
01 00 00 00 B4 01 00 00 7C 02 00 00 D4 01 00 00
72 02 00 00 7F 02 00 00 87 02 00 00 8A 02 00 00
98 02 00 00 9B 02 00 00 9E 02 00 00 A2 02 00 00
AD 02 00 00 B1 02 00 00 B5 02 00 00 CC 02 00 00
E0 02 00 00 F4 02 00 00 0F 03 00 00 23 03 00 00
26 03 00 00 2A 03 00 00 3F 03 00 00 47 03 00 00
4F 03 00 00 57 03 00 00 5F 03 00 00 65 03 00 00
6A 03 00 00 73 03 00 00 02 00 00 00 04 00 00 00
07 00 00 00 0A 00 00 00 0B 00 00 00 0C 00 00 00
0D 00 00 00 0E 00 00 00 0F 00 00 00 11 00 00 00
05 00 00 00 05 00 00 00 00 00 00 00 06 00 00 00
06 00 00 00 54 02 00 00 08 00 00 00 06 00 00 00
5C 02 00 00 09 00 00 00 06 00 00 00 64 02 00 00
0F 00 00 00 08 00 00 00 00 00 00 00 10 00 00 00
08 00 00 00 64 02 00 00 10 00 00 00 08 00 00 00
6C 02 00 00 02 00 05 00 13 00 00 00 02 00 05 00
15 00 00 00 07 00 03 00 17 00 00 00 02 00 04 00
01 00 00 00 02 00 06 00 16 00 00 00 03 00 05 00
18 00 00 00 04 00 04 00 01 00 00 00 06 00 04 00
01 00 00 00 06 00 01 00 12 00 00 00 06 00 02 00
12 00 00 00 06 00 03 00 12 00 00 00 06 00 00 00
19 00 00 00 02 00 00 00 01 00 00 00 04 00 00 00
00 00 00 00 03 00 00 00 00 00 00 00 92 03 00 00
8D 03 00 00 01 00 01 00 01 00 00 00 7D 03 00 00
04 00 00 00 70 10 03 00 00 00 0E 00 05 00 01 00
02 00 00 00 82 03 00 00 2C 00 00 00 12 00 13 01
80 00 35 10 28 00 62 01 02 00 22 02 06 00 70 10
04 00 02 00 1A 03 14 00 6E 20 07 00 32 00 0C 02
6E 20 06 00 02 00 0C 02 1A 03 00 00 6E 20 07 00
32 00 0C 02 6E 20 05 00 02 00 0C 02 6E 10 08 00
02 00 0C 02 6E 20 02 00 21 00 D8 00 00 01 8E 00
28 D7 0E 00 01 00 00 00 00 00 00 00 01 00 00 00
01 00 00 00 01 00 00 00 05 00 00 00 01 00 00 00
09 00 0B 20 63 68 61 72 61 63 74 65 72 20 00 06
3C 69 6E 69 74 3E 00 01 43 00 0C 43 61 73 74 69
6E 67 2E 6A 61 76 61 00 01 49 00 01 4C 00 02 4C
43 00 09 4C 43 61 73 74 69 6E 67 3B 00 02 4C 49
00 02 4C 4C 00 15 4C 6A 61 76 61 2F 69 6F 2F 50
72 69 6E 74 53 74 72 65 61 6D 3B 00 12 4C 6A 61
76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 3B 00
12 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69
6E 67 3B 00 19 4C 6A 61 76 61 2F 6C 61 6E 67 2F
53 74 72 69 6E 67 42 75 69 6C 64 65 72 3B 00 12
4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 79 73 74 65
6D 3B 00 01 56 00 02 56 4C 00 13 5B 4C 6A 61 76
61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 00 06
61 70 70 65 6E 64 00 06 61 73 63 53 74 72 00 06
61 73 63 69 69 20 00 06 63 68 72 53 74 72 00 04
6D 61 69 6E 00 03 6F 75 74 00 07 70 72 69 6E 74
6C 6E 00 08 74 6F 53 74 72 69 6E 67 00 01 00 07
0E 00 08 01 00 07 0E 5A 01 22 0D 4D 00 02 17 14
17 00 02 00 02 00 00 18 01 18 00 81 80 04 D4 03
01 09 EC 03 0E 00 00 00 00 00 00 00 01 00 00 00
00 00 00 00 01 00 00 00 1A 00 00 00 70 00 00 00
02 00 00 00 0A 00 00 00 D8 00 00 00 03 00 00 00
07 00 00 00 00 01 00 00 04 00 00 00 03 00 00 00
54 01 00 00 05 00 00 00 09 00 00 00 6C 01 00 00
06 00 00 00 01 00 00 00 B4 01 00 00 01 20 00 00
02 00 00 00 D4 01 00 00 01 10 00 00 04 00 00 00
54 02 00 00 02 20 00 00 1A 00 00 00 72 02 00 00
03 20 00 00 02 00 00 00 7D 03 00 00 05 20 00 00
01 00 00 00 8D 03 00 00 00 20 00 00 01 00 00 00
92 03 00 00 00 10 00 00 01 00 00 00 A4 03 00 00

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

In his book, *Decompiling Android*, Godfrey Nolan further identifies individual “sections” of a sample .dex file. Nolan identifies a “class\_data\_item” structure within the “data” section (see Chart 1, above) as containing dex bytecode for static fields (class variables), instance fields (instance variables) and pointers to direct method (constructor, static and private methods) and virtual method (all other methods) bytecode. Nolan further identifies a “code\_item” structure within the “data” section of the file as containing the actual dex bytecode necessary for execution. The other metadata fields described in Chart 1 provide links, lists and pointers that relate classes and their methods to the bytecode in the data section.<sup>20</sup> The data structures for bytecode areas in the data section of a .dex file are illustrated in the code snippets below:

```
platform_dalvik-master\libdex\DexFile.hi

/*
 * Direct-mapped "class_def_item".
 */
structDexClassDef {
    u4classIdx;      /* index into typelds for this class */
    u4accessFlags;
    u4superclassIdx; /* index into typelds for superclass */
    u4interfacesOff; /* file offset to DexTypeList */
    u4sourceFileIdx; /* index into stringlds for source file name */
    u4annotationsOff; /* file offset to annotations_directory_item */
    u4classDataOff;  /* file offset to class_data_item */
    u4staticValuesOff; /* file offset to DexEncodedArray */
};

/*
 * Direct-mapped "code_item".
 *
 * The "catches" table is used when throwing an exception,
 * "debugInfo" is used when displaying an exception stack trace or
 * debugging. An offset of zero indicates that there are no entries.
 */
structDexCode {
    u2registersSize;
    u2insSize;
```

---

<sup>20</sup> *Id.*

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

u2outsSize;
u2triesSize;
u4debugInfoOff;    /* file offset to debug info stream */
u4insnsSize;        /* size of the insns array, in u2 units */
u2insns[1];
/* followed by optional u2 padding */
/* followed by try_item[triesSize] */
/* followed by uleb128 handlersSize */
/* followed by catch_handler_item[handlersSize] */
};

```

In the above code, classDataOff within the DexClassDef structure points to a class\_data\_item within a .dex file data section, while insns[1] within the DexCode structure points to “instruction” bytecode for method execution.

**Dex Metadata**

As previously described, the data area of a .dex file provides the bytecode used in class and method execution. Most of the remaining sections of a .dex file, as summarized in Chart 1, above, provide metadata used by the Dalvik VM to find variables and methods in one class (module) that are referenced in another class (module). When .dex files are loaded or “mapped” into memory, these metadata fields will form the basis of the “module information table” characteristic of a code server that provides module information between the classes (modules) in an application (multi-module program). Basically, metadata in a .dex file consists of pointers to class\_data\_items and code\_items found in the .dex file’s data area. Chart 2, below, summarizes the types of pointers found in the various metadata sections of a .dex file.

---

Chart 2. Dex File Metadata Field Pointers

---

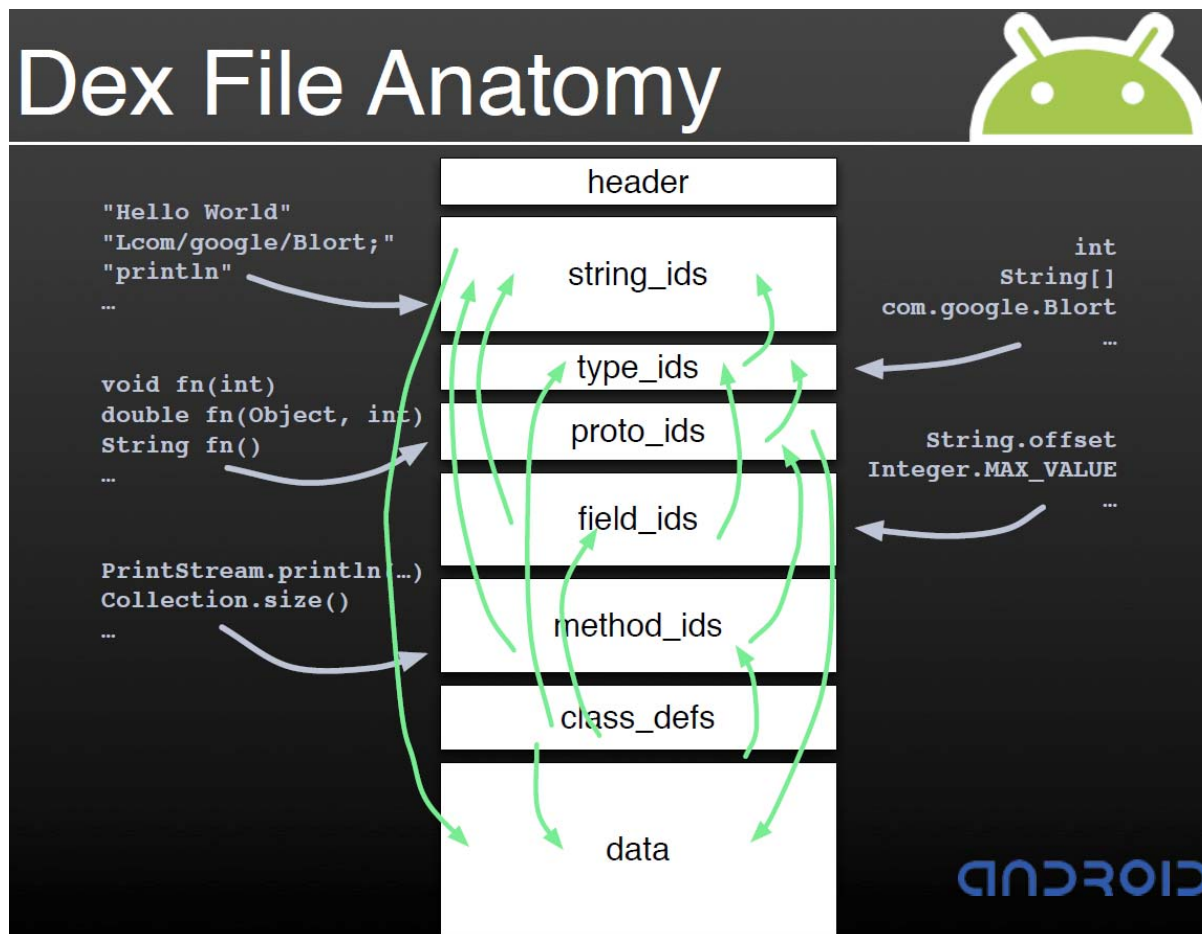
<b>Name</b>	<b>Pointer (Linkage) Description</b>
Header	Defines sizes and pointers into the remaining sections.
string_ids	Contains pointers to strings (class names, method names, variable names) found in the data section.
type_ids	Contains pointers to string names for data types (classes, arrays, or primitives).
proto_ids	Dalvik VM uses proto_ids along with type_ids to assemble method_ids. proto_ids

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Chart 2. Dex File Metadata Field Pointers

Name	Pointer (Linkage) Description
	contain pointers to string_ids (for short description of method parameters), pointers to type_ids (for method return type) and pointers into data section for parameter list.
field_ids	Contains pointers to type_ids (for class name and field type) and pointers to string_ids (for field name).
method_ids	Contains pointers to type_ids (for class names), proto_ids (for method prototype) and string_ids (for method name).
class_defs	Contains pointers to class data.

The figure below<sup>21</sup> shows the types of pointers found in the various metadata sections in Chart 2.



<sup>21</sup> Dan Bornstein, “Dalvik VM Internals”, Presented at Google I/O Session, 2008, <https://sites.google.com/site/io/dalvik-vm-internals>; slides available at <https://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The data structures for metadata fields of a .dex file are illustrated in the following code snippets:

```
platform_dalvik-master\libdex\DexFile.h1

/*
 * Direct-mapped "header_item" struct.
 */
structDexHeader {
    u1magic[8];          /* includes version number */
    u4checksum;          /* Adler32 checksum */
    u1signature[kSHA1DigestLen]; /* SHA-1 hash */
    u4fileSize;          /* length of entire file */
    u4headerSize;        /* offset to start of next section */
    u4endianTag;
    u4linkSize;
    u4linkOff;
    u4mapOff;
    u4stringIdsSize;
    u4stringIdsOff;
    u4typeIdsSize;
    u4typeIdsOff;
    u4protoIdsSize;
    u4protoIdsOff;
    u4fieldIdsSize;
    u4fieldIdsOff;
    u4methodIdsSize;
    u4methodIdsOff;
    u4classDefsSize;
    u4classDefsOff;
    u4dataSize;
    u4dataOff;
};

/*
 * Direct-mapped "string_id_item".
 */
structDexStringId {
    u4stringDataOff;     /* file offset to string_data_item */
};

/*
 * Direct-mapped "type_id_item".
 */
structDexTypeId {
    u4descriptorIdx;     /* index into stringIds list for type descriptor */
};
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

/*
 * Direct-mapped "proto_id_item".
 */
structDexProtoId {
    u4shortyIdx;      /* index into stringIds for shorty descriptor */
    u4returnTypeIdx;  /* index into typeIds list for return type */
    u4parametersOff;  /* file offset to type_list for parameter types */
};

/*
 * Direct-mapped "field_id_item".
 */
structDexFieldId {
    u2classIdx;      /* index into typeIds list for defining class */
    u2typeIdx;       /* index into typeIds for field type */
    u4nameIdx;       /* index into stringIds for field name */
};

/*
 * Direct-mapped "method_id_item".
 */
structDexMethodId {
    u2classIdx;      /* index into typeIds list for defining class */
    u2protoIdx;      /* index into protoIds for method prototype */
    u4nameIdx;       /* index into stringIds for method name */
};

/*
 * Direct-mapped "class_def_item".
 */
structDexClassDef {
    u4classIdx;      /* index into typeIds for this class */
    u4accessFlags;
    u4superclassIdx; /* index into typeIds for superclass */
    u4interfacesOff; /* file offset to DexTypeList */
    u4sourceFileIdx; /* index into stringIds for source file name */
    u4annotationsOff; /* file offset to annotations_directory_item */
    u4classDataOff;  /* file offset to class_data_item */
    u4staticValuesOff; /* file offset to DexEncodedArray */
};

```

**BOOTCLASSPATH and CLASSPATH**

In the Accused Systems, .class files, or .dex files, or .apk files, or .jar files, or .odex files, or archives containing .class files and/or .dex files that are used by an application are stored in the BOOTCLASSPATH and/or CLASSPATH repositories. The BOOTCLASSPATH is a list of the jars/apk from which classes can be loaded or “mapped”, in addition to the main apk/jar that is

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

loaded or “mapped”. The Android system typically has five jars in its base BOOTCLASSPATH: core.jar, ext.jar, framework.jar, android.policy.jar, and services.jar. These can all be found in /system/framework. If an application has a dependency on a .jar file beyond that of the base five jars, then that .jar file will be appended to the BOOTCLASSPATH for that application’s apk.<sup>22</sup>

**Initialization of the Zygote on System Boot**<sup>23</sup>

After the Linux Kernel loads, init.rc is parsed and Android’s built-in services are started. One of the native services started is /system/bin/app\_process. The command from init.rc that kicks off app\_process is service zygote /system/bin/app\_process -Xzygote /system/bin --zygote -start-system-server. The app\_process function calls AndroidRuntime.start(), passing it the parameters com.android.internal.os.ZygoteInit and start-system-server. Code in AndroidRuntime.cpp gets called because runtime is of type AppRuntime and AppRuntime inherits from, or extends, AndroidRuntime. AndroidRuntime.start() starts the Dalvik VM, then calls ZygoteInit.main().<sup>ii</sup>

ZygoteInit.main() first registers the zygote socket (the zygote process listens to a socket for incoming commands, and on receiving a new command, spawns a new process as requested).<sup>iii</sup> Then Zygote initialization loads core classes using preload(). This in turn calls preloadClasses(), whose purpose is to load preloaded-classes. It does this by reading the contents of the file named “preloaded-classes”. Preloaded classes are loaded by class.cpp. Application apk classes are also loaded by class.cpp.

---

<sup>22</sup> <http://code.google.com/p/smali/wiki/DeodexInstructions>.

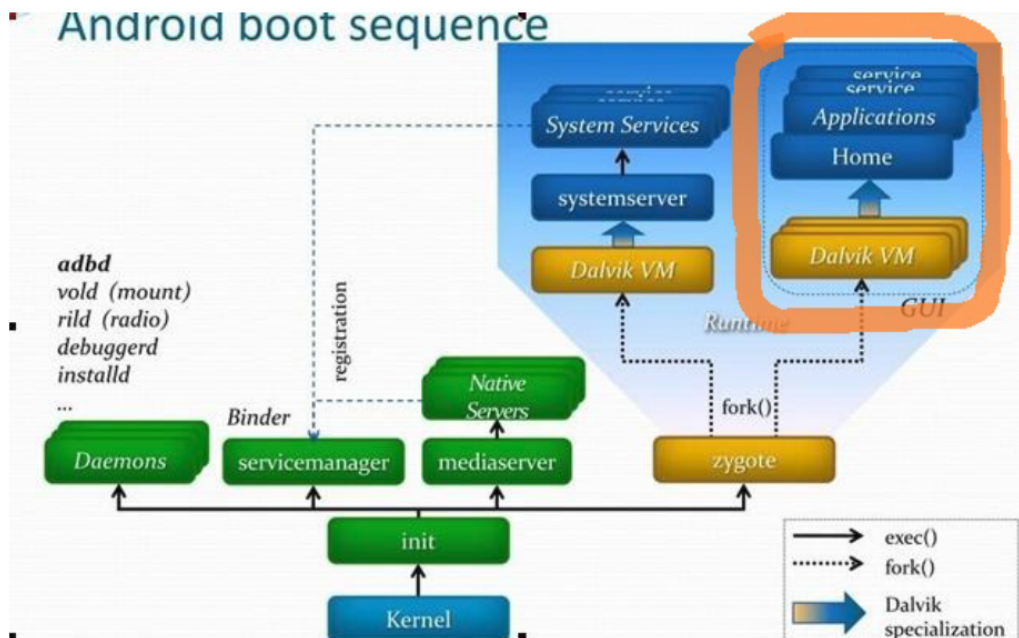
<sup>23</sup> <http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf>;  
<http://www.youtube.com/watch?v=G-36noTCaiA>

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****Fork a Zygote to Create Another Dalvik VM on Application Launch**

When an application is launched, SystemServer signals Zygote to spawn a Dalvik VM by forking a new process. The child process now gets a pre-warmed up Dalvik VM in which to run – all the common Android libraries are already mapped (loaded to shared memory). This process of launching an application may be performed automatically.

After SystemServer is forked (see the diagram below<sup>24</sup>), function `runSelectLoopMode()` is called. This is a `while(true)` loop which establishes a `ZygoteConnection` with the zygote socket and waits for commands on it. When a command is received, `ZygoteConnection.runOnce()` is called.<sup>iv</sup> `ZygoteConnection.runOnce()` calls `Zygote.forkAndSpecialize()` which calls a native function to do the fork.<sup>v</sup> Thus, like in the case of SystemServer, a child process is created which has a pre-warmed up Dalvik VM for itself.<sup>25</sup>

## Dalvik VM

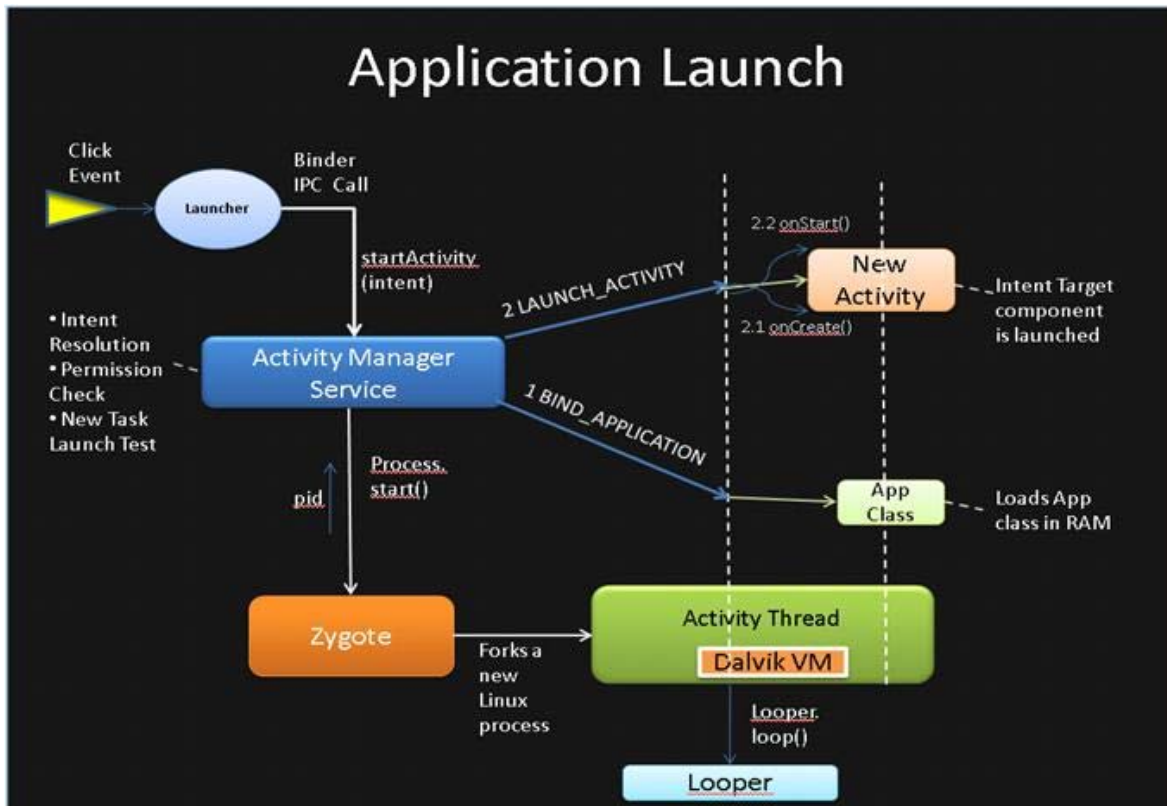


<sup>24</sup> Taras Leskiv, "Memory Management Fundamentals for Android apps," Presented at NRAvo (April 3, 2013) available at <http://www.slideshare.net/tarasleskiv/android-memory-fundamentals>.

<sup>25</sup> <http://stackoverflow.com/questions/9153166/understanding-android-zygote-and-dalvikvm>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****Launching an Application**

The launching of an application, including a widget, is generally illustrated in the diagram below.<sup>26</sup>



Launching an application ends up in the app\_main.cpp code (as when cold-booting Zygote) but the (classname) path is followed.<sup>vi</sup>

When runtime.start is launched, code in AndroidRuntime.cpp gets called because runtime is of type AppRuntime and AppRuntime inherits from, or extends, AndroidRuntime. In AndroidRuntime.cpp, function AndroidRuntime::startVm calls JNI\_CreateJavaVM to initialize the Dalvik VM.<sup>vii</sup> In Jni.cpp, function JNI\_CreateJavaVM calls dvmStartup to continue

<sup>26</sup> <http://coltf.blogspot.ca/p/android-os-processes-and-zygote.html>;  
[http://1.bp.blogspot.com/\\_cga7sfO2vms/TOHGY\\_i5KFI/AAAAAAAAAEko/bBFy5j9U\\_6g/s1600/app+launch+summary.jpg](http://1.bp.blogspot.com/_cga7sfO2vms/TOHGY_i5KFI/AAAAAAAAAEko/bBFy5j9U_6g/s1600/app+launch+summary.jpg).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

initialization of the VM.<sup>viii</sup> In Init.cpp, function dvmStartup calls dvmClassStartup and several more functions to begin loading or “mapping” classes into Dalvik VM memory space.<sup>ix</sup>

**Loading or Mapping Classes**

The BOOTCLASSPATH and CLASSPATH directories hold the metadata that is used to populate the “module information table” that includes “module information” in the form of linkage information for .class files and/or .dex files and/or .apk files and/or .odex files and/or .jar files and/or archives containing .class and/or .dex files. Each class represents a “module” in a “multi-module program”.<sup>x</sup>

In Class.cpp, function dvmClassStartup, as called from Init.cpp, creates hash table gDvm.loadedClasses used for tracking classes (modules) that have been loaded or “mapped” into memory, then calls processClassPath to process bootstrap class path, open specified .dex or .jar files, and possibly run them through the optimizer.<sup>xi</sup> In Class.cpp, function processClassPath, as called from dvmClassStartup, calls prepareCpe to prepare a list of jar and dex class path entries.<sup>xii</sup> In Class.cpp, function prepareCpe, as called from processClassPath, calls dvmJarFileOpen to get .dex files.<sup>xiii</sup>

In JarFile.cpp, function dvmJarFileOpen, as called from prepareCpe, calls dvmDexFileOpenFromFd to map optimized .dex files into memory.<sup>xiv</sup>

In DvmDex.cpp, function dvmDexFileOpenFromFd, as called from dvmJarFileOpen, calls dexFileParse and allocateAuxStructures to create and populate class and method connections.<sup>xv</sup>

First, function dexFileParse<sup>xvi</sup>, as called from dvmDexFileOpenFromFd, returns a data structure of type DexFile. This data structure returns pointers to the metadata and data sections in the source .dex file being loaded or “mapped” into memory:

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

platform_dalvik-master\libdex\DexFile.h

/*
 * Structure representing a DEX file.
 *
 * Code should regard DexFile as opaque, using the API calls provided here
 * to access specific structures.
 */
structDexFile {
/* directly-mapped "opt" header */
constDexOptHeader* pOptHeader;

/* pointers to directly-mapped structs and arrays in base DEX */
constDexHeader* pHeader;
constDexStringId* pStringIds;
constDexTypeId* pTypeId;
constDexFieldId* pFieldIds;
constDexMethodId* pMethodIds;
constDexProtoId* pProtoIds;
constDexClassDef* pClassDefs;
constDexLink* pLinkData;

/*
 * These are mapped out of the "auxillary" section, and may not be
 * included in the file.
 */
constDexClassLookup* pClassLookup;
constvoid* pRegisterMapPool; // RegisterMapClassPool

/* points to start of DEX file data */
constu1* baseAddr;

/* track memory overhead for auxillary structures */
intoverhead;

/* additional app-specific data structures associated with the DEX */
//void* auxData;
};

```

Then function `allocateAuxStructures`,<sup>xvii</sup> as called from `dvmDexFileOpenFromFd`, uses the `DexFile` structure as input and returns a data structure of type `DvmDex`. This data structure holds the metadata and data sections as captured from the `.dex` file being loaded or “mapped” into memory:

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```
platform_dalvik-master\vm\DvmDex.h
```

```
/*
 * Some additional VM data structures that are associated with the DEX file.
 */
structDvmDex {
/* pointer to the DexFile we're associated with */
DexFile*      pDexFile;

/* clone of pDexFile->pHeader (it's used frequently enough) */
constDexHeader* pHeader;

/* interned strings; parallel to "stringIds" */
structStringObject** pResStrings;

/* resolved classes; parallel to "typeIds" */
structClassObject** pResClasses;

/* resolved methods; parallel to "methodIds" */
structMethod**      pResMethods;

/* resolved instance fields; parallel to "fieldIds" */
/* (this holds both InstField and StaticField) */
structField**       pResFields;

/* interface method lookup cache */
structAtomicCache* pInterfaceCache;

/* shared memory region with file contents */
boolisMappedReadOnly;
MemMappingmemMap;

    jobject dex_object;

/* lock ensuring mutual exclusion during updates */
    pthread_mutex_t  modLock;
};
```

Function `allocateAuxStructures` creates in memory table of class and method connections and populates the structure with metadata from the .dex file. The resulting resource pointer table(s) form the “module information table”:

```
pDvmDex->pResStrings = (structStringObject**)
pDvmDex->pResClasses = (structClassObject**)
pDvmDex->pResMethods = (struct Method**)
pDvmDex->pResFields = (struct Field**)
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****Boot Completion and Automatic Application/Widget Startup**

System Server is the first application started by the Zygote, and it continues to live on as a separate process from its parent. The System Server initializes each system service it houses and registers it with the previously started Service Manager. One of the services it starts, the Activity Manager, ends its initialization by sending an intent of type Intent, CATEGORY\_HOME. This starts the Launcher application, which then displays the Android Home screen.<sup>27</sup>

The code below shows the Activity Manager issuing the CATEGORY\_HOME intent and exemplary xml code for the Home screen launcher that receives the intent broadcast:

```
android-platform_frameworks_base-
57ea96a\services\java\com\android\server\am\ActivityManagerService.java
```

```
boolean startHomeActivityLocked() {
    Intent intent = new Intent(
        mTopAction,
        mTopData != null ? Uri.parse(mTopData) : null);
    intent.setComponent(mTopComponent);
    if (mFactoryTest != SystemServer.FACTORY_TEST_LOW_LEVEL) {
        intent.addCategory(Intent.CATEGORY_HOME);
    }
}
```

```
Launcher2-master\AndroidManifest.xml
<application
    android:name="com.android.launcher2.LauncherApplication"
    android:label="@string/application_name"
    android:icon="@mipmap/ic_launcher_home"
    android:hardwareAccelerated="true"
    android:largeHeap="@bool/config_largeHeap"
    android:supportsRtl="true">
    <activity
        android:name="com.android.launcher2.Launcher"
        android:launchMode="singleTask"
        android:clearTaskOnLaunch="true"
        android:stateNotNeeded="true"
        android:theme="@style/Theme"
        android:windowSoftInputMode="adjustPan"
        android:screenOrientation="nosensor">
```

---

<sup>27</sup> *Embedded Android*, by Karim Yaghmour, O'Reilly (2013).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

<intent-filter>
<actionandroid:name="android.intent.action.MAIN"/>
<categoryandroid:name="android.intent.category.HOME"/>
<categoryandroid:name="android.intent.category.DEFAULT"/>
<categoryandroid:name="android.intent.category.MONKEY"/>
</intent-filter>
</activity>

```

There are several types of applications that start automatically upon boot completion: (i) the Home screen launcher triggered by CATEGORY\_HOME intent; (ii) Home screen widgets embedded within Home screen; and applications, widgets and services triggered by BOOT\_COMPLETED intent. Automatically-started application automatically trigger the Zygote to spawn a virtual machine, automatically load or “map” their classes, automatically populate the “module information table”, automatically send a request for module information from the module information table, and receive a response from the module information from the module information table, and/or automatically execute modules.

**Automatic startup of Home screen Launcher triggered by CATEGORY\_HOME intent**

As described above, Application Manager triggers the Launcher application to display the Home screen view. The default launcher class is found in com.android.launcher.Launcher or com.android.launcher2.Launcher. Either of these packages is packed into Launcher.apk.

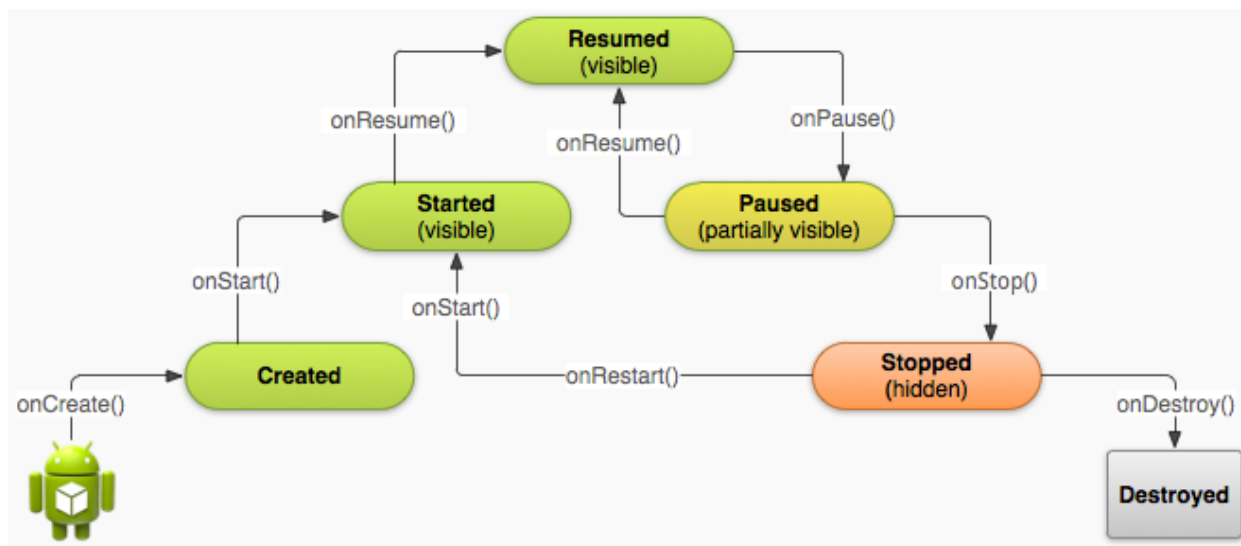
The Launcher is an application like any other. It is packed into an .apk file, Launcher.apk, which is loaded or “mapped” and populated into the resource pointer table(s), and the Launcher runs like any other application. The Launcher contains an activity (display of and user interaction with the Home screen) that conforms to the application “activity lifecycle”, and an examination of the Launcher.java source code<sup>28</sup> reveals the characteristic “callback methods”

---

<sup>28</sup> [https://android.googlesource.com/platform/packages/apps/Launcher2/+/\\_master/src/com/android/launcher2/Launcher.java](https://android.googlesource.com/platform/packages/apps/Launcher2/+/_master/src/com/android/launcher2/Launcher.java).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

of an application activity, such as onCreate, onStart, onResume, onPause, onStop, etc.,<sup>xviii</sup> as shown in the lifecycle illustration below.<sup>29</sup>

**Automatic startup of Home screen widgets embedded within the Home screen**

App Widgets are miniature application views (referred to as “widgets” in the user interface) that can be embedded in other applications (such as the Home screen) and receive periodic updates. An application component that is able to hold other App Widgets is called an App Widget host.<sup>30</sup>

Widgets embedded in the Home screen automatically start during the onCreate phase of Launcher lifecycle. The following code shows an exemplary Launcher.java starting the embedded AppWidgetManager and AppWidgetHost:

Launcher2-master\src\com\android\launcher2\Launcher.java

```

protected void onCreate(Bundle savedInstanceState) {
    mAppWidgetManager = AppWidgetManager.getInstance(this);
    mAppWidgetHost = new LauncherAppWidgetHost(this, APPWIDGET_HOST_ID);
    mAppWidgetHost.startListening();
}

```

<sup>29</sup> <http://developer.android.com/training/basics/activity-lifecycle/starting.html>.

<sup>30</sup> <http://developer.android.com/guide/topics/appwidgets/index.html>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

AppWidgetHost<sup>xix</sup> starts listening for AppWidgetManager to tell it when to load or “map” or update widgets, which AppWidgetManager does by broadcasting the APPWIDGET\_UPDATE intent in conjunction with AppWidgetService:

```
android-platform_frameworks_base-
57ea96a\core\java\android\appwidget\AppWidgetManager.java

/**
 * Sent when it is time to update your AppWidget.
 *
 * <p>This may be sent in response to a new instance for this AppWidget provider
having
 * been instantiated, the requested {@link AppWidgetProviderInfo#updatePeriodMillis
update interval}
 * having lapsed, or the system booting.
 */
public static final String ACTION_APPWIDGET_UPDATE =
"android.appwidget.action.APPWIDGET_UPDATE";

android-platform_frameworks_base-
57ea96a\services\java\com\android\server\AppWidgetService.java

void sendUpdateIntentLocked(Provider p, int[] appWidgetIds) {
if (appWidgetIds != null && appWidgetIds.length > 0) {
    Intent intent = new
Intent(AppWidgetManager.ACTION_APPWIDGET_UPDATE);
    intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, appWidgetIds);
    intent.setComponent(p.info.provider);
    mContext.sendBroadcast(intent);
  }
}
```

As an example, the “Wiktionary” widget is a widget that may be configured for automatic display on the Home screen. The Wiktionary widget displays a Wiktionary “Word of the day”.<sup>31</sup>

The Wiktionary widget’s AndroidManifest.xml file is configured to listen for the APPWIDGET\_UPDATE intent broadcast message, as shown in the following code:

<sup>31</sup> <http://android-developers.blogspot.com/2009/04/introducing-home-screen-widgets-and.html>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Wiktionary widget AndroidManifest.xml<sup>32</sup>

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.wiktionary"
    android:versionCode="1"
    android:versionName="1.0">

    <!-- Broadcast Receiver that will process AppWidget updates -->
    <receiverandroid:name=".WordWidget"android:label="@string/widget_name">
        <intent-filter>
            <actionandroid:name="android.appwidget.action.APPWIDGET_UPDATE"/>
        </intent-filter>
        <meta-
dataandroid:name="android.appwidget.provider"android:resource="@xml/widget_word"/
>
    </receiver>
```

When it receives the APPWIDGET\_UPDATE intent message, WordWidget.java invokes its onUpdate method which starts a service that pushes the quote of the day to the Home screen, as shown in the following code. Note that the WordWidget class “extends”, or “inherits from” AppWidgetProvider class:

Wiktionary WordWidget.java<sup>33</sup>

```
/**
 * Define a simple widget that shows the Wiktionary "Word of the day." To build
 * an update we spawn a background {@link Service} to perform the API queries.
 */
publicclassWordWidgetextendsAppWidgetProvider{
    @Override
    publicvoid onUpdate(Context context,AppWidgetManager appWidgetManager,
        int[] appWidgetIds){
        // To prevent any ANR timeouts, we perform the update in a service
        context.startService(newIntent(context,UpdateService.class));
    }

    publicstaticclassUpdateServiceextendsService{
        @Override
        publicvoid onStart(Intent intent,int startId){
            // Build the widget update for today
            RemoteViews updateViews = buildUpdate(this);

            // Push update for this widget to the home screen
```

---

<sup>32</sup> *Id.*

<sup>33</sup> *Id.*

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

ComponentName thisWidget =newComponentName(this,WordWidget.class);
AppWidgetManager manager =AppWidgetManager.getInstance(this);
manager.updateAppWidget(thisWidget, updateViews);
}

```

Widget modules (classes) are loaded or “mapped” and populated into the resource pointer table(s) along with Launcher Home screen modules (classes) as discussed in Loading or Mapping Classes, above.

Some devices have their own unique widget managers, which work in the same fashion as described above but place an extra layer between the Home screen Widget Manager and Home screen widget(s).

**Automatic startup of apps, widgets, and services triggered by BOOT\_COMPLETED intent**

System Server is the first application started by the Zygote, and it continues to live on as a separate process from its parent. The System Server initializes each system service it houses and registers it with the previously started Service Manager. One of the services it starts, the Activity Manager, starts the Launcher application (which brings up the Home screen) and then sends the BOOT\_COMPLETED intent.

This intent signals to the rest of the platform that the system has completed boot up<sup>34</sup>, as shown in the following code:

```

android-platform_frameworks_base-
57ea96a\services\java\com\android\server\am\ActivityManagerService.java

final void finishBooting() {
// Tell anyone interested that we are done booting!
    SystemProperties.set("sys.boot_completed", "1");
    broadcastIntentLocked(null, null,
new Intent(Intent.ACTION_BOOT_COMPLETED, null),
null, null, 0, null, null,
        android.Manifest.permission.RECEIVE_BOOT_COMPLETED,
false, false, MY_PID, Process.SYSTEM_UID);

```

---

<sup>34</sup> [http://developer.android.com/reference/android/content/Intent.html#ACTION\\_BOOT\\_COMPLETED](http://developer.android.com/reference/android/content/Intent.html#ACTION_BOOT_COMPLETED)

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Applications that are “listening” for the BOOT\_COMPLETED event may use the event to trigger some activity or action in response. The mechanism by which this happens is controlled by the application’s own AndroidManifest.xml file. If the application’s own AndroidManifest.xml file includes “android.intent.action.BOOT\_COMPLETED”, then the application or widget is automatically launched on boot completion.

Applications that are automatically started by the BOOT\_COMPLETED intent are loaded or “mapped” and populated into resource pointer table(s) as discussed above.

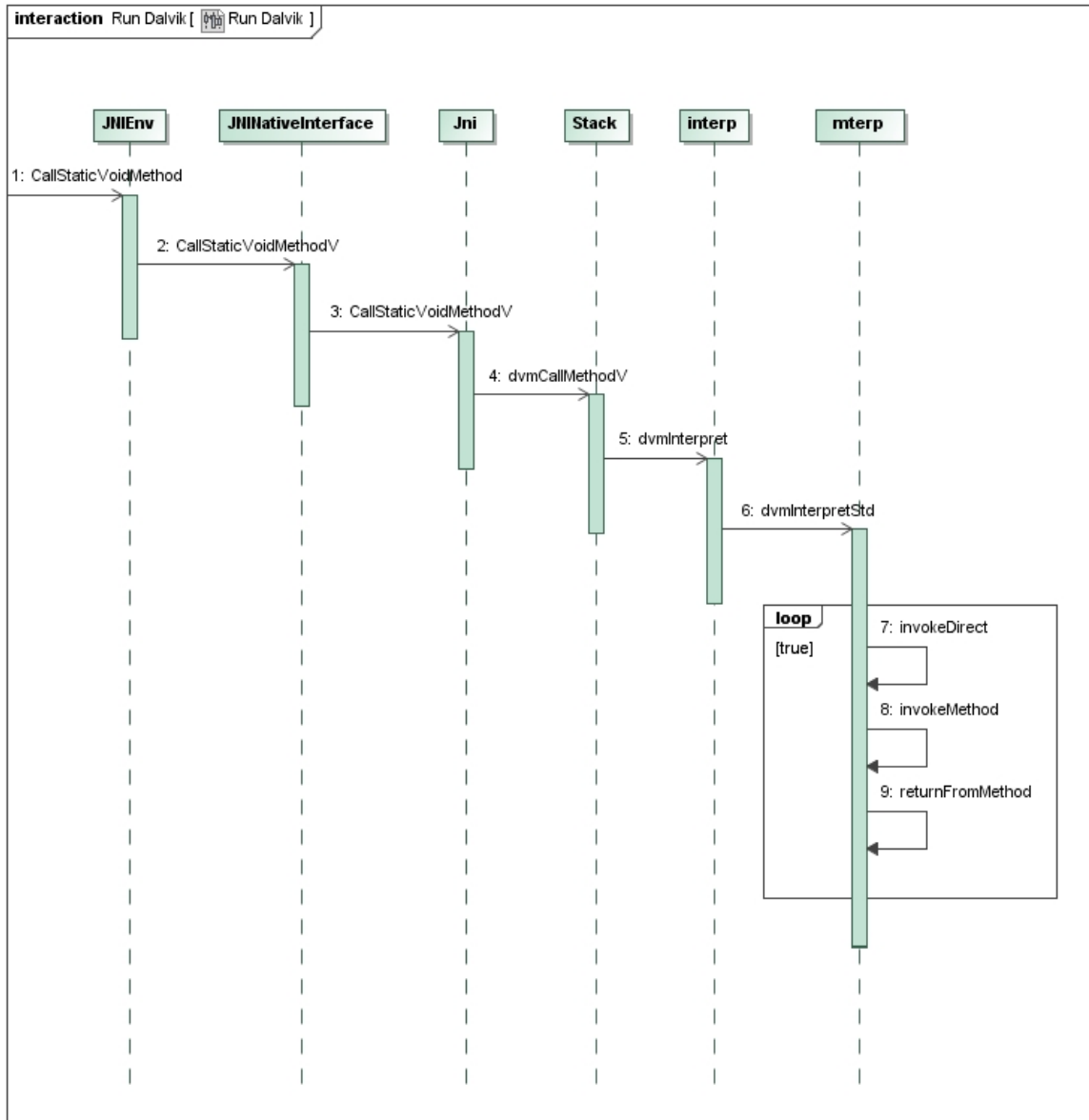
**Application Execution**

After .dex file bytecode is loaded or “mapped” and the “module information table” is created and populated, execution of a module begins at some point thereafter.

Launching an application starts one of the application’s Activities by forking a new clone of the Dalvik VM. In AndroidRuntime.cpp, function AndroidRuntime::start creates the call chain illustrated in the diagram below<sup>35</sup> to launch Android Interpreter code (interp and mterp) to execute the application Activity.

---

<sup>35</sup>“Dalvik virtual machine running analysis,” <http://shyluo.blog.51cto.com/5725845/1229258>, translated as <http://translate.google.com/translate?hl=en&sl=zh-CN&tl=en&u=http%3A%2F%2Fshyluo.blog.51cto.com%2F5725845%2F1229258>).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Entry point to Dalvik VM interpreter is in file `Interp.cpp`.<sup>xx</sup> Dalvik VM interpreter supports three execution modes: (i) portable, (ii) fast, and (iii) JIT. Portable mode can be executed on any platform (*e.g.*, ARM and x86), fast mode is optimized for a specific platform, and JIT<sup>36</sup> is a “just-in-time” compiler. Each is discussed in turn.

<sup>36</sup> Optional JIT capability added with Froyo (Android 2.2). <http://stackoverflow.com/questions/13361293/how-does-minterp-dalvik-vm-organize-its-byte-code-interpret-loop>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Upon initialization (\platform\_dalvik-master\vm\Init.cpp), Dalvik VM parses command line arguments to set global variable gDvm.executionMode to (i) kExecutionModeInterpPortable, (ii) kExecutionModeInterpFast, or (iii) kExecutionModeJit. For Portable Execution Mode, gDvm.executionMode == kExecutionModeInterpPortable. For Fast Execution Mode, gDvm.executionMode == kExecutionModeInterpFast. For JIT Execution Mode, gDvm.executionMode == kExecutionModeJit.

**Portable Execution Mode**

For Portable Execution Mode, gDvm.executionMode == kExecutionModeInterpPortable and interpretation is performed by calling dvmInterpretPortable. Function dvmInterpretPortable is defined in file platform\_dalvik-master\vm\mterp\out\InterpC-portable.cpp.<sup>xxi</sup> The interpreter setup is complete at the end of InterpC-portable.cpp and the interpreter is ready to “fetch and execute first instruction”.

Following the function dvmInterpretPortable in file platform\_dalvik-master\vm\mterp\out\InterpC-portable.cpp is a list of OP CODES. Op codes are assembly language instructions used to execute the application Activity being run. Two op codes, OP\_CONST\_CLASS and GOTO\_TARGET, show calls to functions that locate classes and methods by requesting information from the resource pointer table(s) (module information table).<sup>xxii</sup>

In OP\_CONST\_CLASS, a call is made to dvmDexGetResolvedClass(methodClassDex, ref), which requests information from the resource pointer table(s). If the requested information is not found in the resource pointer table(s), then dvmResolveClass(curMethod->clazz, ref, true) is called. In GOTO\_TARGET, a call is made to dvmDexGetResolvedMethod(methodClassDex, ref) which requests information from the resource pointer table(s). If the requested information

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

is not found in the resource pointer table(s), then `dvmResolveMethod(curMethod->clazz, ref, METHOD_STATIC)` is called.<sup>xxiii</sup>

Functions `dvmDexGetResolvedClass` and `dvmDexGetResolvedMethod` are found in file `platform_dalvik-master\vm\DvmDex.h`.<sup>xxiv</sup> Function `dvmDexGetResolvedClass` requests information from the resource pointer table(s) by Class ID: `pDvmDex->pResClasses[classIdx]`. Function `dvmDexGetResolvedMethod` requests information from the resource pointer table(s) by Method ID: `pDvmDex->pResMethods[methodIdx]`.

**Fast Execution Mode and JIT Execution Mode**

For Fast Execution Mode, `gDvm.executionMode == kExecutionModeInterpFast` and interpretation is performed by calling `dvmMterpStd`. For JIT Execution Mode, `gDvm.executionMode == kExecutionModeJit` and interpretation is also performed by calling `dvmMterpStd`. Function `dvmMterpStd` is defined in file `platform_dalvik-master\vm\mterp\Mterp.cpp`.<sup>xxv</sup>

Function `dvmMterpStd` calls `dvmMterpStdRun`. Function `dvmMterpStdRun` is optimized for a specific platform. The `dvmMterpStdRun` function that runs therefore depends upon the platform. All platforms execute the complete Android op code instruction set but the inner workings of the machine language differ. Implementations of the `dvmMterpStdRun` function are found in the following files:

```
platform_dalvik-master\vm\mterp\out\InterpAsm-armv5te-vfp.Sxxvi
platform_dalvik-master\vm\mterp\out\InterpAsm-armv5te.Sxxvii
platform_dalvik-master\vm\mterp\out\InterpAsm-armv7-a-neon.Sxxviii
platform_dalvik-master\vm\mterp\out\InterpAsm-armv7-a.Sxxix
platform_dalvik-master\vm\mterp\out\InterpAsm-mips.Sxxx
platform_dalvik-master\vm\mterp\out\InterpAsm-x86.Sxxxi
platform_dalvik-master\vm\mterp\out\InterpC-allstubs.cppxxxii
```

Each of the above-listed files contains a set of op codes, including two, `OP_CONST_CLASS`

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

and L\_OP\_INVOKE\_VIRTUAL, that show calls to functions that locate classes and methods by causing searches of the resource pointer table(s) (module information table).<sup>xxxiii</sup> Each of the above-listed files contains platform-specific implementations of OP\_CONST\_CLASS and L\_OP\_INVOKE\_VIRTUAL.

Throughout the above-listed files are conditional statements that are executed only in JIT Execution Mode. These statements are often of the form “**#if** defined(WITH\_JIT)”. Because the JIT instructions are in addition to Fast Execution Mode instructions, JIT is referred to as an extension of the interpreter.<sup>37</sup> The interpreter executes all of the code but in some cases, for parts of method code identified as “hot spots”, those bits of code are also “jitted” to provide for faster execution in subsequent passes.

**Forming the Module Information Table**

As described above, when the Accused Systems run a multi-module program (or, more specifically in the case of the Dalvik VM, a multi-class program), they need to know how the various classes (or modules) link to one another. The Accused Systems (and, in particular, the Dalvik VM) create data structures that include the linkage information for the modules comprising the multi-class program. The Dalvik VM includes several programs that work in concert to create these data structures that include linkage information for a given multi-module program, including linkage information from the .dex file.

The process of forming a data structure that includes linkage information for a multi-class program includes the Dalvik VM creating and populating the resource pointer table(s) as a result of loading or “mapping” the metadata from the .dex file. The resource pointer table(s) (and the corresponding structure for an .odex file) comprises the Module Information Table that includes

---

<sup>37</sup> Srinivas Kothuri, “Android JIT”, <http://www.slideshare.net/SrinivasKothuri/dalvik-jit-31830080>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

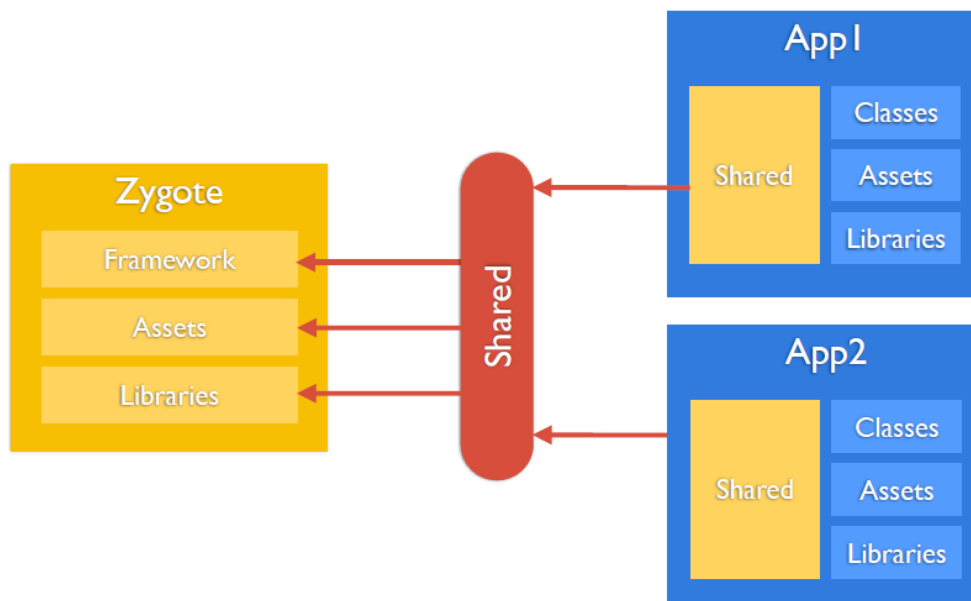
module information for modules in a multi-module program.

**Shared memory**

The Android system shares RAM pages across processes, including in the following ways:<sup>38</sup>

- Each application process is forked from an existing Zygote process. The Zygote process starts when the system boots and loads or “maps” common framework code and resources (such as activity themes). Starting a new application forks the Zygote process, which then loads or “maps” and runs the application’s code in the new process, thereby allowing most of the RAM pages allocated for framework code and resources to be shared across all application processes.
- Most static data is mmap’ed into a process, thereby allowing that same data to be both shared between processes and also paged out when needed.

The figure below<sup>39</sup> depicts these two types of shared memory regions:



<sup>38</sup> <http://developer.android.com/training/articles/memory.html>.

<sup>39</sup> “Memories of Android.” Talk by Romain Guy presented at Devovx, 2013, [https://speakerd.s3.amazonaws.com/presentations/6950a0f032390131a8f91e7b7986a513/Android\\_Memories.pdf](https://speakerd.s3.amazonaws.com/presentations/6950a0f032390131a8f91e7b7986a513/Android_Memories.pdf).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

When .dex files load, class data is loaded or “mapped” into mmap shared memory regions and the resource pointer table(s) (module information table) is populated. When core framework classes are loaded or “mapped”, data is loaded or “mapped” into Zygote shared memory regions.

**Resource pointer table(s) formation**

The Accused Systems form a data structure that includes the linkage information for a set of modules of a given multi-module program. This data structure is referred to by Defendant as a resource pointer table(s), which contains substructures known as pointers:

```
pDvmDex->pResStrings = (structStringObject**)
pDvmDex->pResClasses = (structClassObject**)
pDvmDex->pResMethods = (struct Method**)
pDvmDex->pResFields = (struct Field**)
```

Forming and populating the resource pointer table(s) involves interactions between several programs of the Accused System (in particular, portions of the Dalvik VM) to load or “map” the linkage information from the metadata of a .dex file stored in an apk.

In particular, when the Dalvik VM loads or “maps” an application’s .apk file and parses classes.dex, the dvmDexFileOpenFromFd program calls allocateAuxStructures to create a 4-byte pointer for every reference to a class, method, field, or string constant, for the particular module.<sup>xxxiv</sup> The class, method, field, and string constant pointer meta information comes from .dex file metadata fields. The following is the code that populates the resource pointer table(s):

```
platform_dalvik-master\vm\DvmDex.cpp

/*
 * Create auxillary data structures.
 *
 * We need a 4-byte pointer for every reference to a class, method, field,
 * or string constant.
 */

staticDvmDex* allocateAuxStructures(DexFile* pDexFile)
{
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

DvmDex* pDvmDex;
constDexHeader* pHeader;
u4 stringSize, classSize, methodSize, fieldSize;

    pHeader = pDexFile->pHeader;

    stringSize = pHeader->stringIdsSize * sizeof(structStringObject*);
    classSize = pHeader->typeIdsSize * sizeof(structClassObject*);
    methodSize = pHeader->methodIdsSize * sizeof(structMethod*);
    fieldSize = pHeader->fieldIdsSize * sizeof(structField*);

    u4 totalSize = sizeof(DvmDex) +
        stringSize + classSize + methodSize + fieldSize;

    u1 *blob = (u1 *)dvmAllocRegion(totalSize,
        PROT_READ | PROT_WRITE, "dalvik-aux-structure");
    if ((void *)blob == MAP_FAILED)
    return NULL;

    pDvmDex = (DvmDex*)blob;
    blob += sizeof(DvmDex);

    pDvmDex->pDexFile = pDexFile;
    pDvmDex->pHeader = pHeader;

    pDvmDex->pResStrings = (structStringObject**)blob;
    blob += stringSize;
    pDvmDex->pResClasses = (structClassObject**)blob;
    blob += classSize;
    pDvmDex->pResMethods = (structMethod**)blob;
    blob += methodSize;
    pDvmDex->pResFields = (structField**)blob;

    ALOGV("+++ DEX %p: allocateAux (%d+%d+%d+%d)*4 = %d bytes",
        pDvmDex, stringSize/4, classSize/4, methodSize/4, fieldSize/4,
        stringSize + classSize + methodSize + fieldSize);

    pDvmDex->pInterfaceCache =
    dvmAllocAtomicCache(DEX_INTERFACE_CACHE_SIZE);

    dvmInitMutex(&pDvmDex->modLock);

    return pDvmDex;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****Formation of resource pointer table(s) from dex optimization**<sup>40</sup>

Application code is provided to the Accused Systems in a .jar or .apk file. These are zip archives with some metadata files added. The bytecode cannot be memory-mapped and executed directly from the .zip file because the data is compressed and the start of the file is not guaranteed to be word-aligned. Therefore, the .dex file must be extracted from the zip archive and prepared for execution before it can be used.

Such a “prepared” or “optimized” .dex file (referred to as an “.odex file”) is created in at least three different ways: (i) the Dalvik VM prepares the .dex file “just in time”; (ii) the system installer prepares the .dex file when an application is added; and (iii) the build system prepares the .dex file ahead of time. A core apk may be “pre-optimized” by stripping the classes.dex file from the core .apk file and generating an .odex file.

When the Dalvik VM runs a non-optimized application for the first time, it invokes dex optimization by calling the platform\_dalvik-master\dexopt\OptMain.cpp<sup>xxxv</sup> program. OptMain.cpp is entered at function main.<sup>xxxvi</sup> From there, the following function chain ensues: fromDex → dalvik-master\vm\analysis\DexPrepare.cpp\dvmContinueOptimization → rewriteDex → verifyAndOptimizeClasses → platform\_dalvik-master\vm\oo\Class.cpp\dvmLookupClass.

In file DexPrepare.cpp<sup>xxxvii</sup>, function **verifyAndOptimizeClasses** uses .dex file metadata fields pDexFile->pHeader->classDefsSize and pClassDef->classIdx (refer to Dex Metadata) that is appended to the end of the .odex file. After the .odex file is loaded or “mapped” the appended data populate the resource pointer table(s).

---

<sup>40</sup> This section extracted from source code file platform\_dalvik-master\docs\dexopt.html, also available for viewing at <http://www.netmite.com/android/mydroid/dalvik/docs/dexopt.html>

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

platform_dalvik-master\vm\analysis\DexPrepare.cpp

/*
 * Perform in-place rewrites on a memory-mapped DEX file.
 */
static bool rewriteDex(u1* addr, int len, bool doVerify, bool doOpt,
DexClassLookup** ppClassLookup, DvmDex** ppDvmDex)
{
    DexClassLookup* pClassLookup = NULL;
    u8 prepWhen, loadWhen, verifyOptWhen;
    DvmDex* pDvmDex = NULL;
    bool result = false;
    const char* msgStr = "???";

/*
 * Create the class lookup table. This will eventually be appended
 * to the end of the .odex.
 *
 * We create a temporary link from the DexFile for the benefit of
 * class loading, below.
 */
    pClassLookup = dexCreateClassLookup(pDvmDex->pDexFile);
    if (pClassLookup == NULL)
        goto bail;
    pDvmDex->pDexFile->pClassLookup = pClassLookup;

/*
 * Load all classes found in this DEX file. If they fail to load for
 * some reason, they won't get verified (which is as it should be).
 */
    if (!loadAllClasses(pDvmDex))
        goto bail;
    loadWhen = dvmGetRelativeTimeUsec();

/*
 * Verify and optimize all classes in the DEX file (command-line
 * options permitting).
 */
    verifyAndOptimizeClasses(pDvmDex->pDexFile, doVerify, doOpt);

/*
 * Verify and/or optimize all classes that were successfully loaded from
 * this DEX file.
 */
static void verifyAndOptimizeClasses(DexFile* pDexFile, bool doVerify,
bool doOpt)
{
    u4 count = pDexFile->pHeader->classDefsSize;
    u4 idx;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

for (idx = 0; idx < count; idx++) {
    constDexClassDef* pClassDef;
    constchar* classDescriptor;
    ClassObject* clazz;

    pClassDef = dexGetClassDef(pDexFile, idx);
    classDescriptor = dexStringByTypeIdx(pDexFile, pClassDef->classIdx);

    /* all classes are loaded into the bootstrap class loader */
    clazz = dvmLookupClass(classDescriptor, NULL, false);
    if (clazz != NULL) {
        verifyAndOptimizeClass(pDexFile, clazz, pClassDef, doVerify, doOpt);
    }
}

platform_dalvik-master\vm\oo\Class.cpp

/*
 * Search through the hash table to find an entry with a matching descriptor
 * and an initiating class loader that matches "loader".
 */
ClassObject* dvmLookupClass(constchar* descriptor, Object* loader,
bool unprepOkay)
{
    ClassMatchCriteria crit;
    void* found;
    u4 hash;

    crit.descriptor = descriptor;
    crit.loader = loader;
    hash = dvmComputeUtf8Hash(descriptor);

    LOGVV("threadid=%d: dvmLookupClass searching for '%s' %p",
        dvmThreadSelf()->threadId, descriptor, loader);

    dvmHashTableLock(gDvm.loadedClasses);
    found = dvmHashTableLookup(gDvm.loadedClasses, hash, &crit,
        hashcmpClassByCrit, false);
    dvmHashTableUnlock(gDvm.loadedClasses);

    return (ClassObject*) found;
}

```

**Usage of resource pointer table(s) to provide linkages between classes**

Dalvik VM program execution is performed by an “interpreter” (see Application Execution, above). The interpreter translates dex bytecode to machine language instructions that the device’s CPU can understand. Each instruction the interpreter understands is known as an

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

“op code”. The set of all op codes is known as the “instruction set”. The Dalvik instruction set contains several op codes for executing, or “invoking”, methods. When the interpreter comes to the op code invoke-virtual (execute a virtual method), it needs to know where to find the bytecode that contains the method’s instructions. Each method, however, belongs to a unique class.

To find a method’s bytecode, the interpreter requests module information from the resource pointer table(s). The interpreter receives information from the resource pointer table(s) regarding the linkages between the class invoking the method and the class which contains the method being invoked. The information includes links to the invoked method’s bytecode. Thus, the interpreter can invoke the method and execute its instructions.

The interpreter uses functions `dvmDexGetResolvedClass(methodClassDex, ref)` and `dvmDexGetResolvedMethod(methodClassDex, ref)` to cause searches of the resource pointer table(s). The responses are in the form of **return** `pDvmDex->pResClasses[classIdx]` which provides a struct `ClassObject*` data structure to the interpreter and **return** `pDvmDex->pResMethods[methodIdx]`, which provides a struct `Method*` data structure to the interpreter.

The following code snippets show these functions that are used to request module information from the Module Information Table:

`platform_dalvik-master\vm\DvmDex.h`

```

INLINE structClassObject* dvmDexGetResolvedClass(constDvmDex* pDvmDex,
u4 classIdx)
{
    assert(classIdx < pDvmDex->pHeader->typeIdsSize);
    return pDvmDex->pResClasses[classIdx];
}

```

```

INLINE structMethod* dvmDexGetResolvedMethod(constDvmDex* pDvmDex,
u4 methodIdx)
{
    assert(methodIdx < pDvmDex->pHeader->methodIdsSize);
    return pDvmDex->pResMethods[methodIdx];
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

}

The `dvmInterpretPortable` program forms a collection of data that includes the contents of classes, which in turn include mapped information from the metadata of classes from the `.dex` file. This information is not organized in a haphazard fashion, where it would be of little use. Instead it is stored in data structures, as shown in the following code snippet:

```
platform_dalvik-master\vm\DvmDex.h
/*
 * Some additional VM data structures that are associated with the DEX file.
 */
structDvmDex {
/* pointer to the DexFile we're associated with */
DexFile*      pDexFile;

/* clone of pDexFile->pHeader (it's used frequently enough) */
constDexHeader* pHeader;

/* interned strings; parallel to "stringIds" */
structStringObject** pResStrings;

/* resolved classes; parallel to "typeIds" */
structClassObject** pResClasses;

/* resolved methods; parallel to "methodIds" */
structMethod**      pResMethods;

/* resolved instance fields; parallel to "fieldIds" */
/* (this holds both InstField and StaticField) */
structField**      pResFields;

/* interface method lookup cache */
structAtomicCache* pInterfaceCache;

/* shared memory region with file contents */
boolisMappedReadOnly;
MemMappingmemMap;

    jobject dex_object;

/* lock ensuring mutual exclusion during updates */
    pthread_mutex_t  modLock;
};
```

The data structures formed by `dvmInterpretPortable` include `pResClasses` and `pResMethods`. As shown by the following comments included in those data structures,

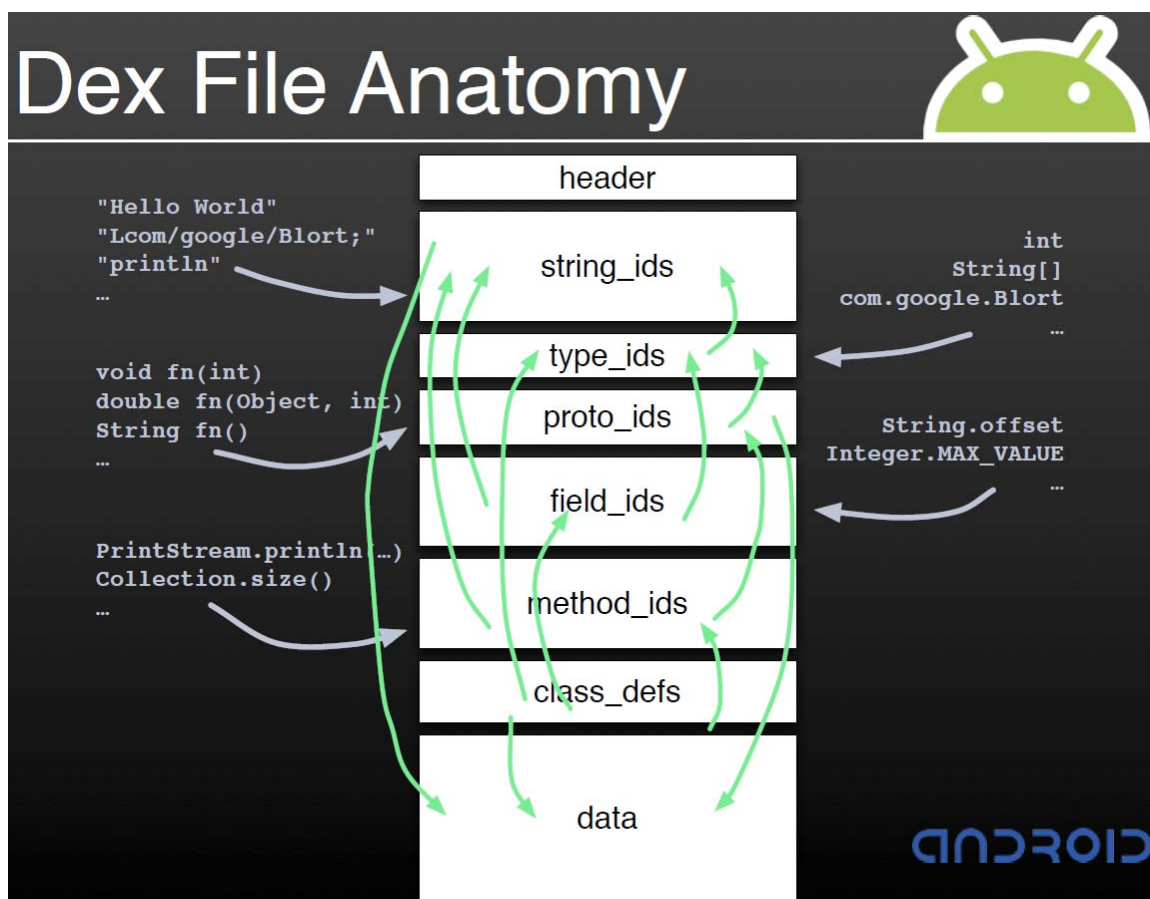
**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

pResClasses includes the same module information as the typeIds .dex file metadata structure, and pResMethods includes the same module information as the methodIds .dex file metadata structure:

```
platform_dalvik-master\vm\DvmDex.h
/* resolved classes; parallel to "typeIds" */
structClassObject** pResClasses;

/* resolved methods; parallel to "methodIds" */
structMethod** pResMethods;
```

The typeIds and methodIds .dex file metadata structures enable the linking together of one or more modules of the second multi-module program to one or more other modules. The figure below<sup>41</sup> illustrates the linkage pointers found in the various .dex file metadata sections.



<sup>41</sup> Dan Bornstein, “Dalvik VM Internals”, Presented at Google I/O Session, 2008, <https://sites.google.com/site/io/dalvik-vm-internals>; <https://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****A note about pointers or links in data structures**

In computer programming, information does not need to physically reside in data structure to be deemed included in that data structure. One of the most common ways to include information in a data structure is by including within that data structure pointers to the information residing in other data structures. The definitive reference on data structures is D. E. Knuth, “The Art of Computer Programming”, vol. 1, (Addison Wesley Longman), first published in the 1960s and most recently revised in 1997. Chapter 2 of that volume is Information Structures, and most of the chapter concerns how to construct and use various sorts of data structures using pointers, which Knuth prefers to call links (not to be confused with linkage information, which is discussed throughout these infringement contentions). Knuth uses the terms “link” and “pointer” interchangeably. In computer science, they mean the same thing:

The introduction of links [*i.e.*, pointers] to other elements is an extremely important idea in computer programming; links are the key to the representation of complex structures. ... Notice also that (3) indicates the top card [in a data structure representing a group of playing cards] by an arrow from “TOP”; here TOP is a link variable, often called a pointer variable, namely a variable whose value is a link. All references to nodes in a program are made directly through link variables.<sup>42</sup>

It is universally accepted to those skilled in the art that “links” in a given data structure, and the contents of what is linked to, are themselves contents of that data structure. Were it otherwise, complex data structures would be unwieldy and impractical. For example, a standard data structure is a “linked list” in which each item in a sequence is stored in a small data structure with a link (pointer) to the next item. By using the links, it is easy to step through the list in sequence. Imagine a linked list containing items A, B, D, E, F, G, and a programmer needs to insert item C between B and D. If the list were stored sequentially in a single storage area, first

---

<sup>42</sup> D. E. Knuth, “The Art of Computer Programming”, vol. 1, (Addison Wesley Longman).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

the program would have to move items D, E, F, and G in order to make room for C. But using a linked list, item C can be placed in any available storage location, and the links adjusted so that C logically is between B and D. In applications that manage long lists of data, these techniques save a great deal of time that would otherwise be spend shuffling list items back and forth to create space for new items and close up space from deleted items.

Linked lists are also an essential technique to save space and avoid data duplication. Imagine an application that maintains a list of items to be delivered, and the addresses to deliver them to. If each address is stored along with the item, in the common case that several items are going to the same address, the address information is duplicated for each item. If the addresses are stored separately, with a link from the item to the address, all the items for a given address can link to the same copy of the address, saving a great deal of storage. Now imagine that one of the recipients has moved, so her address needs to be updated. In the linked scenario, a single update to her address suffices to update all of the packages to be delivered, while if the addresses were stored with each item, they'd have to be updated in each place they occur, taking extra time and introducing a potential source of inconsistent data if the addresses aren't all updated at the same time.

These and other list and pointer management techniques are taught in every undergraduate computer science curriculum, and are well known to any programmer of ordinary skilled in the art.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****The Claim Construction**

The Court in *REC Software USA, Inc. v. Bamboo Solutions Corporation, et al.*, Case No. 2:11-cv-00554-JLR (W.D. Wash.) (the “REC/MSFT Case”), issued a claim construction for the following terms of the asserted claims of the ‘936 patent [REC/MSFT Case Doc. No. 159]:

<b>Term</b>	<b>Construction</b>
<b>“first program that is executing on a computer”</b>	“a set of computer instructions running on a computer that enables the computer to perform a specific operation or operations”
<b>“second multi-module program”</b>	“a set of computer instructions that comprises two or more modules and enables the computer to perform a specific operation or operations”
<b>“code server”</b>	“an identifiable set of computer instructions, different than the first program, which maintains and provides to the first program upon request module information for one or more modules of a multi-module program”
<b>“searching a module information table for module information in response to said request associated with said discrete module”</b>	“in response to a request that includes an identification of the discrete module, the code server searches for module information in the module information table pertaining to the discrete module”
<b>“forming an association of said multi-module program by said first program”</b>	“the first program forms a data structure with information (such as dynamic links) concerning how one or more modules of a second multi-module program link to one or more other modules”

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****The Agreed Terms**

The parties in the REC/MSFT Case agreed on the construction of the following terms of the asserted claims of the '936 patent [REC/MSFT Case Doc. No. 134]:

<b>Term</b>	<b>Agreed Construction</b>
<b>“module”</b>	“a unit of a computer program that is unique and identifiable for purposes of compiling, linking, and loading”
<b>“discrete module”</b>	“a specific module”
<b>“a request associated with said discrete module”</b>	“a request that includes an identification of the discrete module”
<b>“module information”</b>	“information about a module that includes at least a list of references specifically identifying the other modules that the module links to”
<b>“module information table”</b>	“a data structure of the code server for storing module information for multiple modules independent of both the executable modules themselves and the first program.” (“A module information table” is distinct from an “association.”)
<b>“at a point prior to execution time of said multi module program being associated”</b>	“at a point in time before any instruction of the multi-module program being associated is executed”
<b>“second multi-module program which includes an embedded reference to a discrete module”</b>	“plain meaning”

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****The Asserted Claims**

<b>Claim 1</b>
<b>Preamble: Claim Language</b>
“A method of providing information to a first program that is executing on a computer for forming an association for a second multi-module program which includes an embedded reference to a discrete module, said method comprising the steps of:”

This element is satisfied if, applying the claim language as construed: (i) the Accused Systems provide information to a first program that is executing on a computer; (ii) the first program includes the function of forming an association for a second multi-module program; and (iii) the second multi-module program includes an embedded reference to a discrete module.

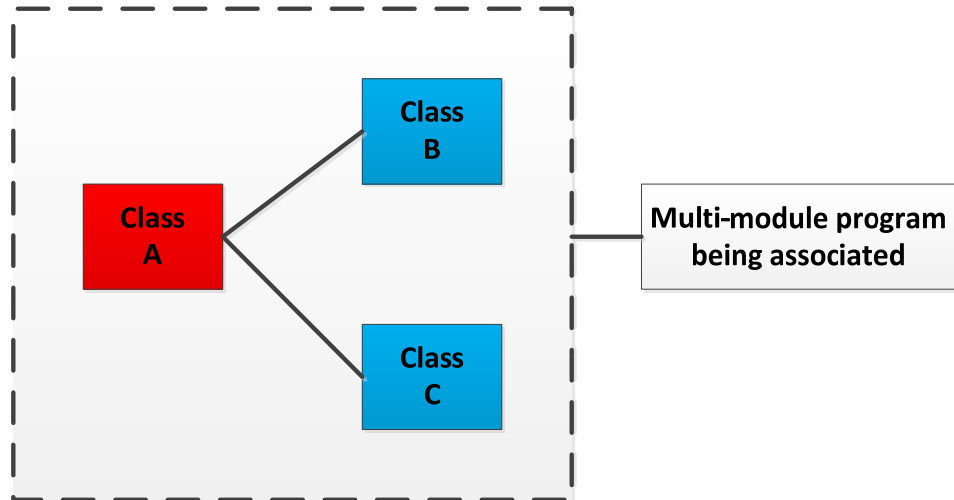
These requirements are addressed in reverse order:

<b>Claim 1: Preamble</b>
“ <b><u>second multi-module program</u></b> which includes an embedded reference to a discrete module”

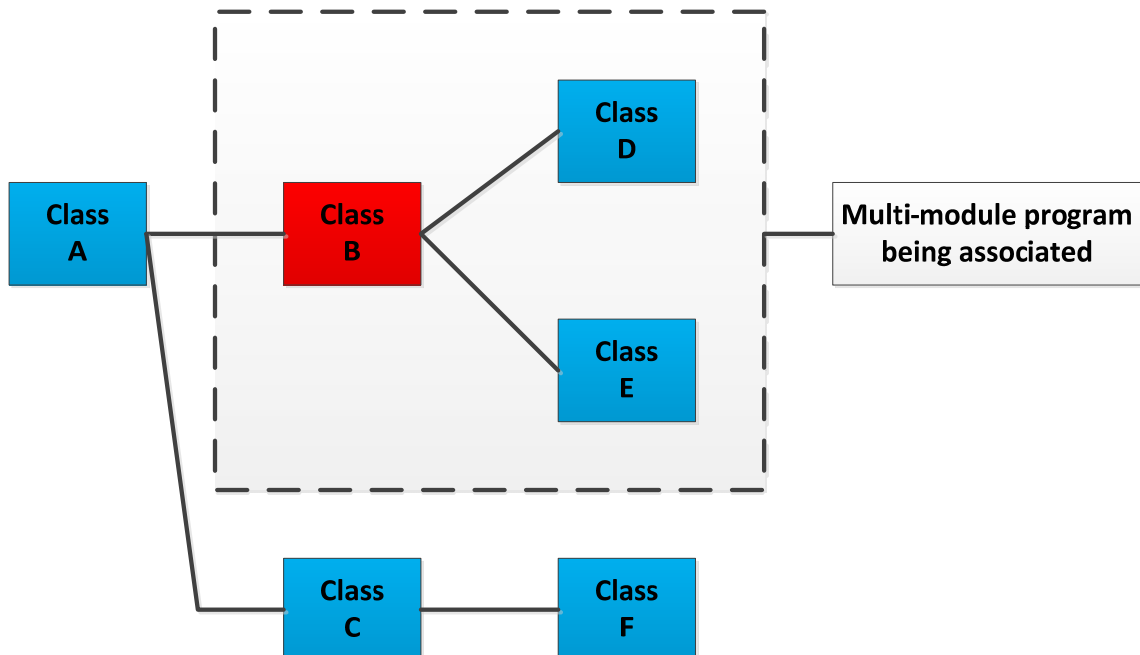
The “second multi-module program” is a .dex program or its corresponding derived optimized .dex program (*i.e.*, an .odex program).<sup>43</sup> A .dex program that is the “second multi-module program” consists of (i) one class found in the .dex file (such class being a “discrete module” as referenced in the claims), and (ii) any class (or classes) that such “discrete module” links to. That is, with respect to a .dex program consisting of classes A-C in the manner depicted in the figure below, if class A is found in the .dex file (and is therefore the claimed “discrete module”), and class A depends upon, or links to, classes B and C (both of which may or may not be found in the .dex file), then the “multi-module program” is the program consisting of classes A, B, and C.

---

<sup>43</sup> Throughout the remainder of this analysis, the use of the term “.dex program” designates, as applicable, either an original .dex program or its corresponding derived .odex program.

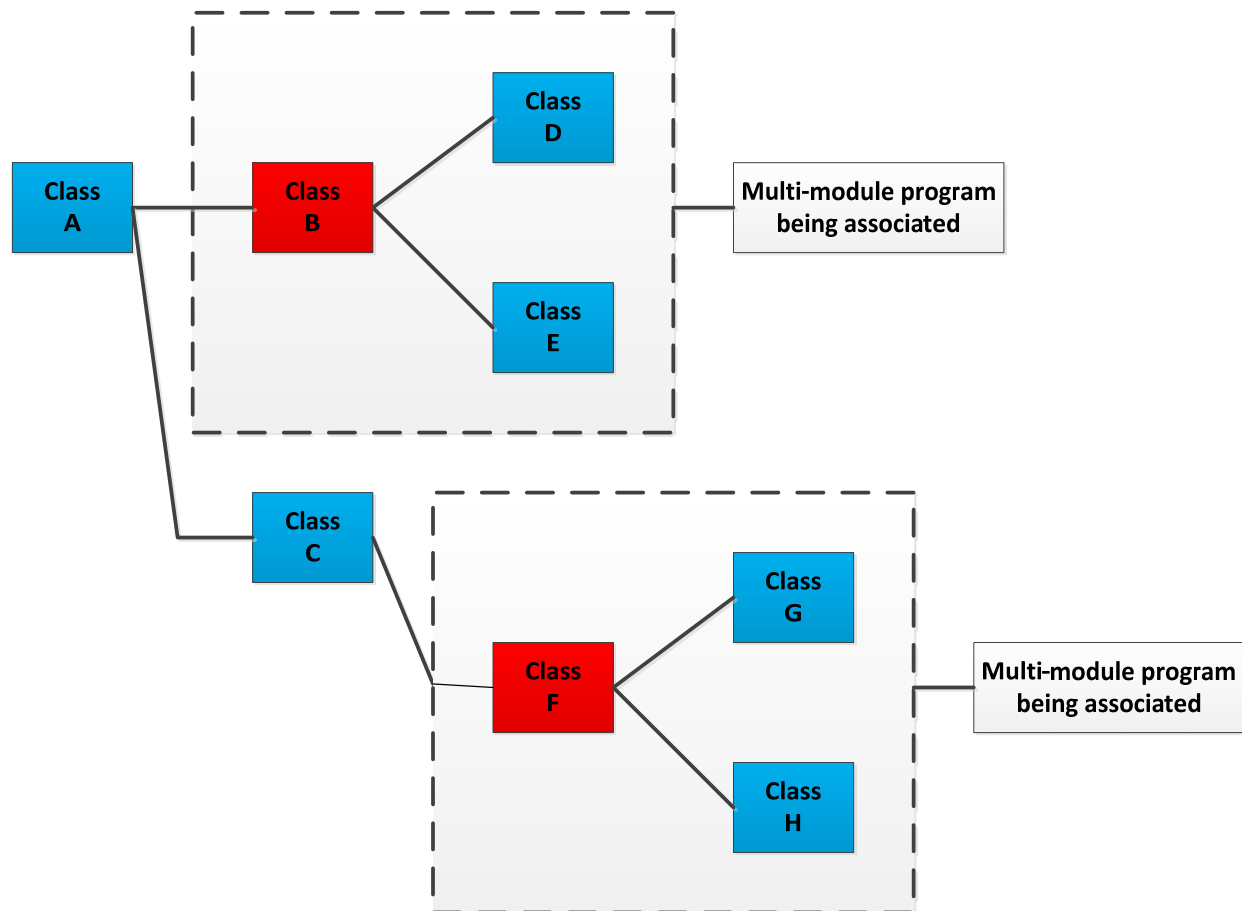
**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Similarly, if a .dex application consists of classes A-F in the manner depicted in the figure below, where class B is the only class that necessarily resides in the .dex file, and class B links to classes D and E, then the “multi-module program” is the program consisting of classes B, D, and E, *i.e.*, class B, the class found in the .dex file, is the “discrete module” referenced by the claims, and classes D and E (both of which may or may not be found in the .dex file) are the classes linked to by the “discrete module”. Classes A, C, and F are not part of this “multi-module program”.



**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Any given .dex application can include several different “multi-module programs”. Each such program would consist of any set of linked classes that begins with a class located in the .dex file. An example is the .dex application consisting of classes A-H in the manner depicted in the figure below, where classes B and F are stored in the .dex file. In this case, the program consisting of classes B, D, and E is a “multi-module program” because class B is the claimed “discrete module” and classes D and E (both of which may or may not be found in the .dex file) are the classes linked to by the “discrete module”. The program consisting of classes F, G, and H is also a “multi-module program” because class F is the claimed “discrete module” and classes G and H (both of which may or may not be found in the .dex file) are the classes linked to by the “discrete module”.



**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Moreover, every .dex application includes at least one “multi-module program” because every .dex program makes use of at least one class from the .dex file, and, as demonstrated below, the program consisting of that class from the .dex file and its linked-to classes meets the requirements of a “multi-module program”.

<b>Claim 1: Preamble</b>
--------------------------

“ <b><u>second</u></b> multi-module program which includes an embedded reference to a discrete module”
--

The term “second” merely distinguishes the “second multi-module program” from the “first program”; that is, the first program and second program are two different programs. The second multi-module program is the program that the operating system is attempting to load or “map” and run, whereas the first program is comprised of components of the operating system itself (including portions of the Dalvik VM) that are working to load or “map” and run the second multi-module program. Accordingly, the first program and second multi-module program are different programs.

<b>Claim 1: Preamble</b>
--------------------------

“second multi- <b><u>module</u></b> program which includes an embedded reference to a discrete module”
--

In the context of Accused Systems, Defendant typically refers to the units that make up a multi-module .dex program as “classes” instead of “modules”. A .dex file that includes a class (*i.e.*, a module) is a particular type of file that includes an additional component. In addition to the executable code found in a typical module, a .dex file also includes metadata that includes information about the module.

In the REC/MSFT Case, the parties agreed to a construction of “module” that reflects the ordinary meaning of the term: “a unit of a computer program that is unique and identifiable for

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

purposes of compiling, linking, and loading”. To demonstrate how a class satisfies this construction, the construction is broken down into its component parts:

**i. a unit of a computer program**

Classes can be combined to form programs (in particular, multi-module programs, or multi class-programs), and a class is therefore a unit of a computer program.

**ii. unique and identifiable**

A class is unique and identifiable if it can be uniquely identified and distinguished from other classes. A fully-qualified class name is the name of the class that includes its package name, such as com.android.launcher2.Launcher or java.awt.Rectangle. A fully-qualified class name uniquely identifies it from other classes.

**iii. compiling**

Compiling is the process of translating source code from one form to another, including, for example, from one language (*e.g.*, Java) into another format (*e.g.*, .class format). Classes are the units of a program that Java code is compiled into, which is translated into .dex format.

**iv. linking**

Linking is the process of resolving dependencies, or linkages, between units of a multi-module program in order to know how the units fit together for execution of the program. Classes are units of a multi-module program that are linked to one another. Indeed, this is evidenced by the classes, which specifies other classes that the given class links to. For example, class source code may include an “import” statement which specifies another package (collection of classes) or class that is linked to (*e.g.*, **import** android.app.Activity).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****v. loading**

Loading is the process of placing a unit of a program from storage into memory for execution. As such, instructions are loaded or “mapped” from the EEPROM or non-volatile storage device so that the CPU can execute the instructions (run the program). “Mapping” makes instructions in non-volatile storage available to CPU for execution by assigning addresses to those instructions that correspond to memory areas in process memory address space.<sup>44</sup> As described above, classes are units of a program that are loaded or “mapped” for execution.

**Claim 1: Preamble**

“second **multi-module program** which includes an embedded reference to a discrete module”

The Court in the REC/MSFT Case construed “second multi module program” as “a set of computer instructions that comprises two or more modules and enables the computer to perform a specific operation or operations”. To demonstrate how the identified “multi-module program” satisfies this construction, the construction is broken down into its component parts.

**i. comprises two or more modules.**

Because a class is a module, and the identified “multi-module program” comprises at least two classes (*i.e.*, the class stored in the .dex file and the class or classes to which the class file stored in the .dex file links), the identified “multi-module program” comprises two or more modules.

**ii. a set of computer instructions**

Because any given class is itself a set of computer instructions (*i.e.*, it consists of an executable module), the two or more classes of the identified “multi-module program being

<sup>44</sup> See *Microsoft Computing Dictionary*, 5th ed. (2002), 315 (defining “load” as “to place information from storage into memory, if it is data, or for execution, if it is code.”).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

associated” necessarily also constitute a set (*i.e.*, a larger set) of computer instructions (see the analysis of “set of computer instructions” set forth below).

**iii. that enables the computer to perform a specific operation or operations**

Each class in the .dex file itself enables the computer to perform a specific operation or operations. Multiple classes are made available for the very reason that multi-class programs can incorporate that particular class and take advantage of the operation or operations it provides.

For example, an exemplary class named Launcher found in a .dex file named classes.dex enables the computer to perform the operation of displaying the Home screen. This is useful for programs or applications that offer a return to Home screen view. Any “multi-module program being associated” that includes this particular class also enables the computer to perform the operation of displaying the Home screen.

That is, because (a) a .dex program enables the computer to perform a specific operation, and (b) the identified multi-module program includes the particular class as a unit of the multi-module program, the multi-module program also enables the computer to perform at least the same specific operation.

Accordingly, the identified “multi-module program” satisfies the requirements of the claim language “second multi-module program”.

<b>Claim 1: Preamble</b>
“second multi-module program which includes an <b><u>embedded reference to a discrete module</u></b> ”

A multi-module program includes an embedded reference to a discrete module if information sufficient to identify the discrete module is built within the program. The identified multi-module programs all have built within them information sufficient to identify the discrete module. The “discrete module” is a class – in particular a class of the multi module program that

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

is stored in the .dex program. The discrete module for which there is an embedded reference within the multi-module program may also reside outside of the multi-module program, such as in a “framework” .dex or .odex file.

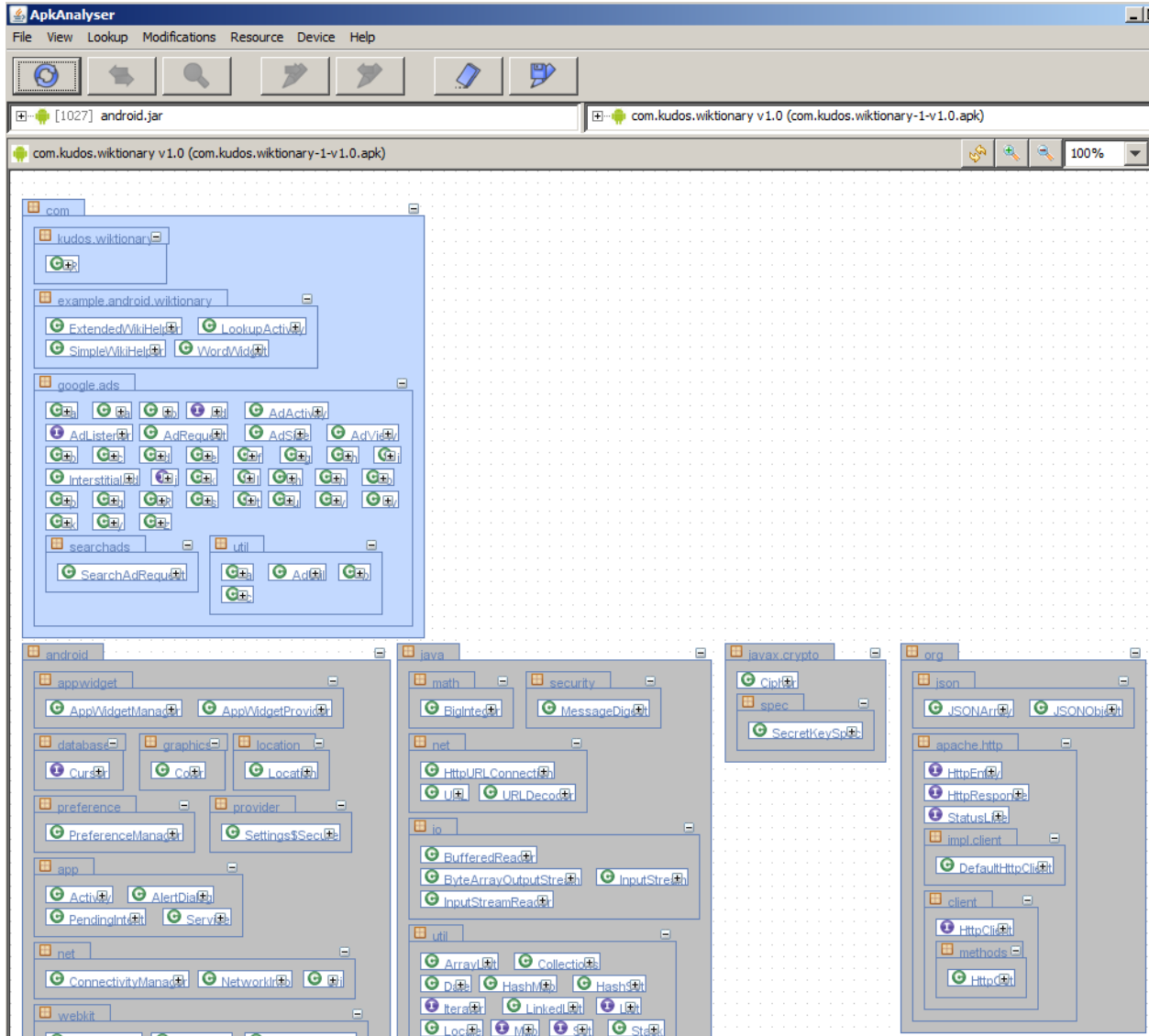
Each class of the identified multi-module program has built with it in the .dex program not only (i) an executable module, but also (ii) metadata that includes information about the executable module. The metadata includes within it information sufficient to identify a particular module. In particular, the metadata includes linkage information of the executable module it accompanies in the same .dex file. The linkage information is sufficient to identify the particular module the metadata accompanies.

By way of example, we can look at typical metadata stored in the .dex program. The metadata built within this .dex program (and hence built within any program that uses this .dex file), includes within it information that identifies a particular class.

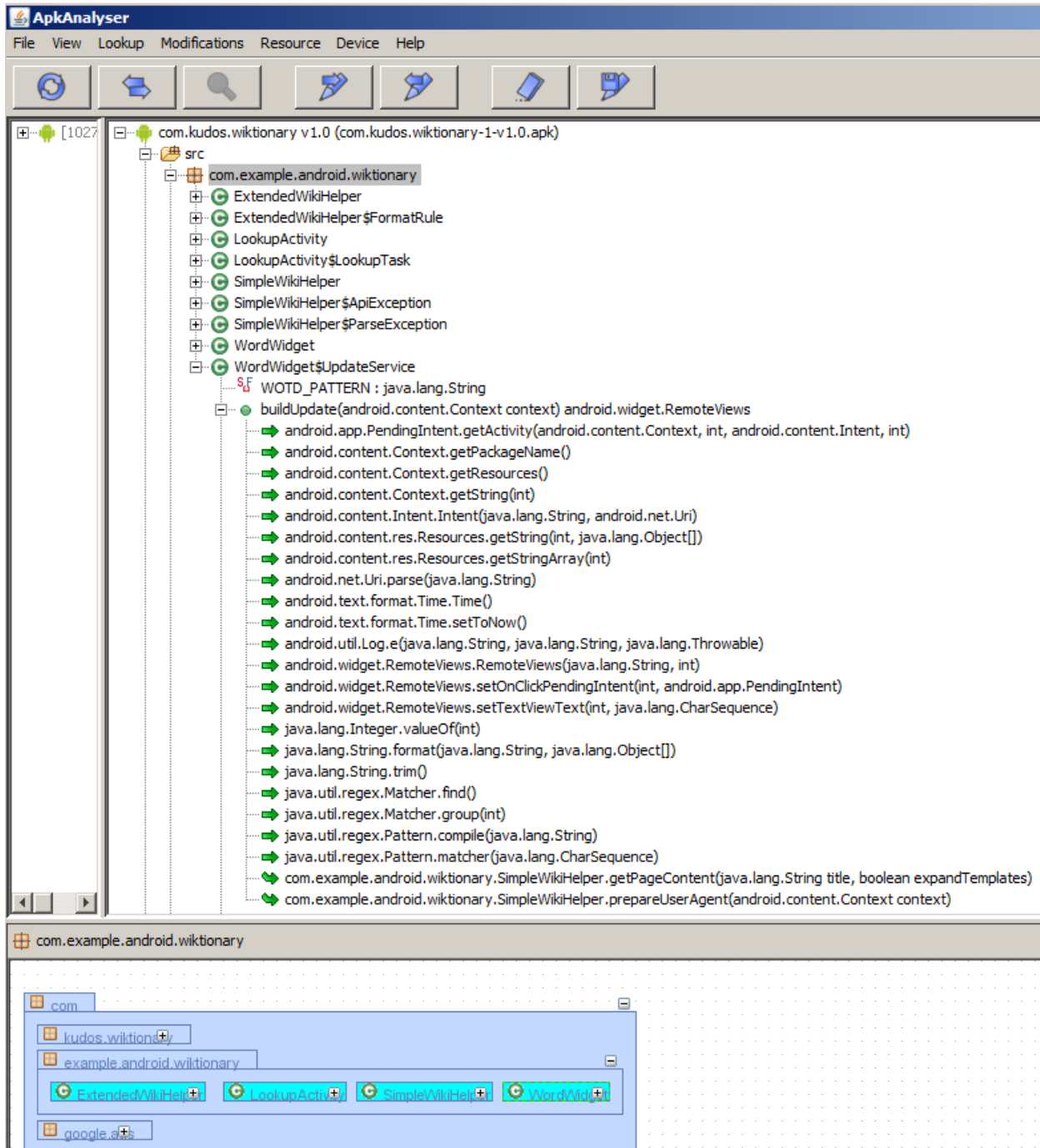
This is illustrated by using the tool ApkAnalyser<sup>45</sup>, which allows the contents of a .dex program to be viewed in a human readable form, as shown in the figure below. The figure shows interlinked class modules for the Wiktionary “Word of the Day” widget discussed above.

---

<sup>45</sup> <http://developer.sonymobile.com/knowledge-base/tools/analyse-your-apks-with-apkanalyser/>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The information sufficient to identify the particular module is identified by the ApkAnalyser tool and presented in graphical format. Classes may be expanded and links to other modules may be viewed. For example, in the code window view of ApkAnalyser shown in the figure below, class WordWidget\$UpdateService, method buildUpdate has embedded references to several modules, including class SimpleWikiHelper.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Accordingly, because the identified multi-module programs include metadata that includes information sufficient to identify the discrete module, the identified multi-module programs include an “embedded reference to a discrete module”.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****Claim 1: Preamble**

“first program that is executing on a computer for forming an association [for a second multi-module program]”

This claim language is satisfied as demonstrated below in connection with Claim 1, Element (e) (“forming an association of said multi-module program by said first program”).

**Claim 1: Preamble**

“A method of providing information to a first program [that is executing on a computer for forming an association]”

This claim language is satisfied if the Accused Systems provide information to a program that the operating system is executing on a computer (*i.e.*, a computing device) for the purpose of forming an association. As explained below in connection with Claim 1, Element (d), the Accused Systems include computer instructions that execute on a computing device for forming an association, and, as also explained below, information is provided to these computer instructions, including information about the referenced modules.

**Claim 1: Preamble: Conclusion**

The Accused Systems satisfy this element.

**Claim 1: Preamble: Doctrine of Equivalents**

In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

<b>Claim 1</b>
<b>Element (a): Claim Language</b>
“receiving in a code server from said first program, at a point prior to execution-time of said multi-module program being associated, a request associated with said discrete module”

This element is satisfied if, applying the claim language as construed, the Accused Systems include: (i) a “code server”; (ii) receiving in the code server from the first program a request associated with the discrete module; and (iii) receiving the request at a point in time before any instruction of the multi-module program being associated is executed.

These requirements are addressed in turn.

<b>Claim 1: Element (a)</b>
“receiving in a <b><u>code server</u></b> from said first program, at a point prior to execution-time of said multi-module program being associated, a request associated with said discrete module”

The code server of the Accused Systems consists of the following components of the Android system: (1) portions of the Dalvik VM together with the metadata obtained from the .dex program stored in BOOTCLASSPATH and/or CLASSPATH directories, (2) the metadata obtained from the `dvmDexFileOpenFromFd` computer instructions, which are responsible for loading or “mapping” the .dex file including its class files and metadata, (3) `allocateAuxStructures` computer instructions, which are responsible for the operation of creating and populating the data structure for a loaded or “mapped” copy of the metadata from the .dex file, and (4) `pDvmDex->pResClasses[classIdx]` and/or `pDvmDex->pResMethods[methodIdx]`.

Applying the construction of the Court in the REC/MSFT Case, the Accused Systems include a code server if they include “an identifiable set of computer instructions, different than the first program, which maintains and provides to the first program upon request module information for one or more modules of a multi-module program”. To demonstrate how the

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Accused Systems satisfy this construction, the construction is broken down into its component parts:

**i. an identifiable set of computer instructions**

A thing is identifiable if it can be identified, or distinguished, from among other like things. Any computer instructions are, by their very nature, identifiable. As explained in the Background section, computer instructions can exist as machine code (strings of 1's and 0's) or as virtual code, which is a more human readable form that represents the same string of 1's and 0's. Each set of instructions consists of a finite set of 1's and 0's (or a finite set of virtual code). Accordingly, one set of computer instructions is identifiable from another set of computer instructions because the 1's and 0's (or virtual code) that correspond to the former instructions can be distinguished from the 1's and 0's (or virtual code) that correspond to the latter instructions. Each of the components that make up the "code server" itself comprises an identifiable set of computer instructions, and collectively the components of the code server form another identifiable set of computer instructions.

A "set" is "a collection of articles designed for use together" or "a number, group, or combination of things of similar nature, design or function."<sup>46</sup> The instructions that make up each component of the code server are a collection of computer instructions designed for use together and a group of similar things (*i.e.*, computer instructions). Therefore the instructions that make up each component of the code server are a set of computer instructions. Moreover, the components of the code server (*i.e.*, the sets of computer instructions that make up the code server) are, collectively, a collection of computer instructions designed for use together and a group of similar things (*i.e.*, sets of computer instructions).

---

<sup>46</sup> See Dictionary.com, "set", <http://dictionary.reference.com/browse/set>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Accordingly, the components of the code server are collectively an identifiable set of computer instructions. Each component of the set of computer instructions that makes up the code server is addressed:

- a. portions of the Dalvik Virtual Machine together with the metadata obtained from the .dex program stored in BOOTCLASSPATH and/or CLASSPATH directories

Portions of the Dalvik VM together with the metadata obtained from the dex program stored in BOOTCLASSPATH and/or CLASSPATH repository is a software component. It is comprised of an identifiable set of computer instructions. An .apk file that includes a .dex file is installed onto a computer not by adding a new hardware component, but by adding a software installation. Likewise, the Dalvik VM that includes a .dex file is provided onto a computer not by adding a new hardware component, but by including a software installation. That is, by adding software (in particular, the Dalvik VM), which is comprised of computer instructions (1's and 0's).

- b. the metadata obtained from the dvmDexFileOpenFromFd computer instructions, which are responsible for loading or “mapping” the .dex file including its .class files and metadata

The operation of loading or “mapping” the metadata obtained from the .dex program is also comprised of an identifiable set of computer instructions. In particular, the dvmDexFileOpenFromFd function comprises an identifiable set of computer instructions to carry out the operation of loading or “mapping” the metadata obtained from the .dex program regarding linkage information provided from the .dex program for a class.

- c. allocateAuxStructures computer instructions, which are responsible for the operation of creating and populating the data structure for a loaded or “mapped” copy of the metadata from the .dex file

The operation of creating and allocating the data structures from the metadata obtained from the .dex program is also comprised of an identifiable set of computer instructions. In

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

particular, the `allocateAuxStructures` function comprises an identifiable set of computer instructions to carry out the operation of creating and populating the data structure for a mapped copy of the metadata from the loaded `.dex` program.

d. `pDvmDex->pResClasses[classIdx]` and/or `pDvmDex->pResMethods[methodIdx]`  
`pDvmDex->pResClasses[classIdx]` and `pDvmDex->pResMethods[methodIdx]` are an identifiable set of computer instructions.

**ii. different than the first program**

The claims recite both a “code server” and a “first program”. As construed by the Court in the REC/MSFT Case, the code server must be an identifiable set of computer instructions “different than” the first program. A set of computer instructions is “different than” another set of computer instructions if the two sets of computer instructions are distinct from one another.

As discussed in connection with the analysis of Claim 1, Element (e), the “first program” is the portion of the Dalvik VM that is responsible for forming the association as a result of `dvmInterpretPortable` program in file `platform_dalvik-master/vm/mterp/out/InterpC-portable.cpp`. Accordingly, this requirement is met if `dvmInterpretPortable` program is a distinct set of computer instructions from the identified “code server” of the Accused Systems, i.e., (1) the metadata obtained from the `.dex` program stored in `BOOTCLASSPATH` and/or `CLASSPATH` directories, (2) the metadata obtained from the `dvmDexFileOpenFromFd` computer instructions, (3) `allocateAuxStructures` computer instructions, and (4) `pDvmDex->pResClasses[classIdx]` and/or `pDvmDex->pResMethods[methodIdx]`.

The computer instructions of the `dvmInterpretPortable` program are distinct from the computer instructions comprising the `dvmDexFileOpenFromFd`, `allocateAuxStructures`, and `pDvmDex->pResClasses[classIdx]` and/or `pDvmDex->pResMethods[methodIdx]`. We know this to be true because these sets of computer instructions perform entirely different operations. For

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

example, the `dvmDexFileOpenFromFd` and `allocateAuxStructures` locate and load or “map” metadata obtained from `.dex` files and populate corresponding data tables using the metadata, while `dvmInterpretPortable` program forms the association for the multi-module program to be executed.

Accordingly, the code server is different than (*i.e.*, a different set of computer instructions than) the first program.

**iii. maintains ... module information for one or more modules of a multi-module program**

The Accused Systems include a “code server” that “maintains ... module information for one or more modules of a multi-module program” if the identified “code server” includes a data structure with module information for one or more modules of a multi-module program. These two requirements are addressed in turn.

a. a data structure with module information

A data structure with module information is an organized collection of data that stores information about a module that includes at least a list of references specifically identifying the other modules that the module links to.<sup>47</sup> The portions of the Dalvik VM of the Accused Systems comprise an organized collection of data that stores information about a module that includes at least a list of references specifically identifying the other modules that the module

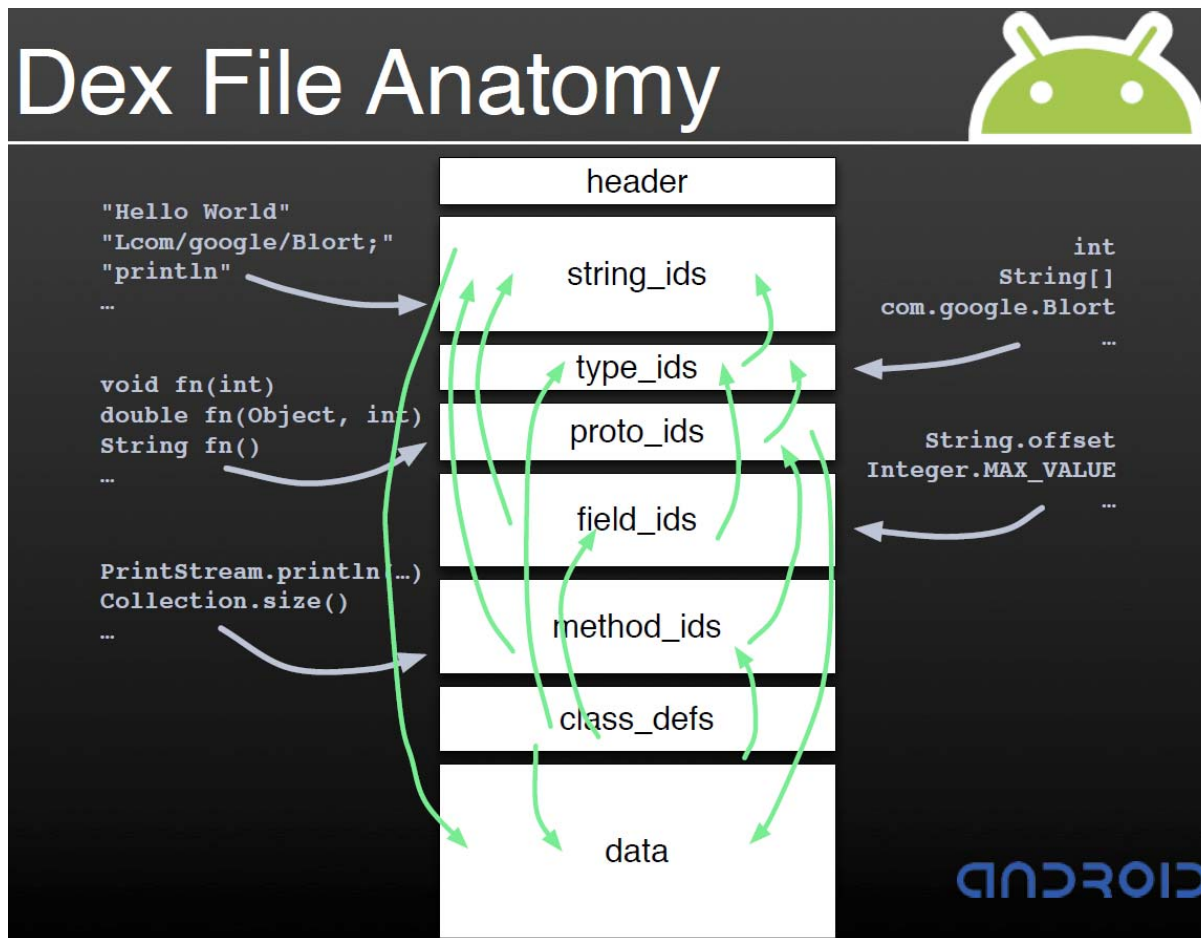
---

<sup>47</sup> See, e.g., *McGraw-Hill Dictionary of Electronics and Computer Technology* (1984) (defining “data structure” as “a collection of data components that are constructed in a regular and characteristic way.”); *Wiley Electrical and Electronics Engineering Dictionary* (2004) (defining “data structure” as “an organizational scheme, such as a record or array, that can be applied to data to facilitate interpreting the data of performing operations on it.”); *Microsoft Computer Dictionary* (4th ed. 1999) (defining “data structure” as “an organizational scheme, such as a record or array, that can be applied to data to facilitate interpreting the data or performing operations on it.”); see also Agreed Constructions, construing “module information” as “information about a module that includes at least a list of references specifically identifying the other modules that the module links to”.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

links to. In particular, the portions of the Dalvik VM maintains the information in an organized manner.

The image below<sup>48</sup> illustrates the .dex file maintaining module information in an organized manner:



The .dex files stored in the non-volatile memory consist of data, including not only the executable code of the module, but also organized metadata, that when loaded or “mapped” is in the form of a data structure (*i.e.*, resource pointer table) corresponding to that module, which likewise is an organized collection of data.

<sup>48</sup> Dan Bornstein, “Dalvik VM Internals”, Presented at Google I/O Session, 2008, <https://sites.google.com/site/io/dalvik-vm-internals>; <https://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The metadata of a .dex file loaded or “mapped” into the Dalvik VM includes information about the modules of from the .dex file. This information includes, for example, (1) `method_ids` that are identifiers for all methods referred to by this file, whether defined in the file or not; this list must be sorted, where the defining type (by `type_id` index) is the major order, method name (by `string_id` index) is the intermediate order, and method prototype (by `proto_id` index) is the minor order; the list must not contain any duplicate entries, and (2) `class_defs` that are classes that must be ordered such that a given class’s superclass and implemented interfaces appear in the list earlier than the referring class; furthermore, it is invalid for a definition for the same-named class to appear more than once in the list (*see* Chart 3, “Dex File Structure”, below).

Accordingly, the metadata is organized in the .dex file and in the corresponding loaded or “mapped” data structure of the resource pointer table(s).

---

Chart 3. Dex File Structure<sup>49</sup>

---

<b>Name</b>	<b>Format</b>	<b>Description</b>
header	header_item	the header
string_ids	string_id_item[]	string identifiers list. These are identifiers for all the strings used by this file, either for internal naming (e.g., type descriptors) or as constant objects referred to by code. This list must be sorted by string contents, using UTF-16 code point values (not in a locale-sensitive manner), and it must not contain any duplicate entries.
type_ids	type_id_item[]	type identifiers list. These are identifiers for all types (classes, arrays, or primitive types) referred to by this file, whether defined in the file or not. This list must be sorted by <code>string_id</code> index, and it must not contain any duplicate entries.
proto_ids	proto_id_item[]	method prototype identifiers list. These are identifiers for all prototypes referred to by this file. This list must be sorted in return-type (by <code>type_id</code> index) major order, and then by arguments (also by <code>type_id</code> index). The list must not contain any duplicate entries.
field_ids	field_id_item[]	field identifiers list. These are identifiers for all fields referred to by this file, whether defined in the file or not. This list must be sorted, where the defining type (by <code>type_id</code> index) is the major order, field name (by <code>string_id</code> index) is the intermediate order, and type (by <code>type_id</code> index) is the minor order. The list must not contain any duplicate entries.

---

<sup>49</sup> <http://source.android.com/devices/tech/dalvik/dex-format.html>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**Chart 3. Dex File Structure<sup>49</sup>

Name	Format	Description
method_ids	method_id_item[]	method identifiers list. These are identifiers for all methods referred to by this file, whether defined in the file or not. This list must be sorted, where the defining type (by type_id index) is the major order, method name (by string_id index) is the intermediate order, and method prototype (by proto_id index) is the minor order. The list must not contain any duplicate entries.
class_defs	class_def_item[]	class definitions list. The classes must be ordered such that a given class's superclass and implemented interfaces appear in the list earlier than the referring class. Furthermore, it is invalid for a definition for the same-named class to appear more than once in the list.
data	ubyte[]	data area, containing all the support data for the tables listed above. Different items have different alignment requirements, and padding bytes are inserted before each item if necessary to achieve proper alignment.
link_data	ubyte[]	data used in statically linked files. The format of the data in this section is left unspecified by this document. This section is empty in unlinked files, and runtime implementations may use it as they see fit.

The image below<sup>50</sup> illustrates a binary dex file shown in hexadecimal format with metadata and data sections highlighted.

<sup>50</sup> Hexadecimal view of a classes.dex file with sections highlighted, after Godfrey Nolan, *Decompiling Android*, Chapter 3 (Apress 2012).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

64	65	78	0A	30	33	35	00	62	8B	44	18	DA	A9	21	CA
9C	4F	B4	C5	21	D7	77	BC	2A	18	4A	38	0D	A2	AA	FE
50	04	00	00	70	00	00	00	78	56	34	12	00	00	00	00
00	00	00	00	A4	03	00	00	00	00	00	00	70	00	00	00
0A	00	00	00	D8	00	00	00	00	00	00	00	01	00	00	00
03	00	00	00	54	01	00	00	09	00	00	00	6C	01	00	00
01	00	00	00	B4	01	00	00	7C	02	00	00	D4	01	00	00
72	02	00	00	7F	02	00	00	87	02	00	00	8A	02	00	00
98	02	00	00	9B	02	00	00	9E	02	00	00	A2	02	00	00
AD	02	00	00	B1	02	00	00	B5	02	00	00	CC	02	00	00
E0	02	00	00	F4	02	00	00	0F	02	00	00	23	03	00	00
26	03	00	00	2A	03	00	00	3F	03	00	00	47	03	00	00
4F	03	00	00	57	03	00	00	5F	03	00	00	65	03	00	00
6A	03	00	00	73	03	00	00	02	00	00	00	04	00	00	00
07	00	00	00	0A	00	00	00	0B	00	00	00	0C	00	00	00
0D	00	00	00	0E	00	00	00	0F	00	00	00	11	00	00	00
05	00	00	00	05	00	00	00	06	00	00	00	06	00	00	00
06	00	00	00	54	02	00	00	08	00	00	00	06	00	00	00
5C	02	00	00	09	00	00	00	06	00	00	00	64	02	00	00
0F	00	00	00	08	00	00	00	00	00	00	00	10	00	00	00
08	00	00	00	64	02	00	00	10	00	00	00	08	00	00	00
6C	02	00	00	02	00	05	00	13	00	00	00	02	00	05	00
15	00	00	00	07	00	00	00	11	00	00	00	02	00	04	00
01	00	00	00	02	00	00	00	15	00	00	00	03	00	05	00
18	00	00	00	04	00	04	00	01	00	00	00	06	00	04	00
01	00	00	00	06	00	01	00	12	00	00	00	06	00	02	00
12	00	00	00	06	00	00	00	00	00	06	00	00	00	00	00
19	00	00	00	02	00	00	00	01	00	00	00	04	00	00	00
00	00	00	00	03	00	00	00	00	00	00	00	92	03	00	00
8D	03	00	00	01	00	01	00	01	00	00	00	7D	03	00	00
04	00	00	00	70	10	03	00	00	00	0E	00	05	00	01	00
02	00	00	00	82	03	00	00	20	00	00	00	12	00	13	01
80	00	35	10	28	00	62	01	02	00	00	22	02	06	00	70
04	00	02	00	1A	03	14	00	6E	20	07	00	32	00	0C	02
6E	20	06	00	02	00	0C	02	1A	03	00	00	6E	20	07	00
32	00	0C	02	6E	20	05	00	02	00	0C	02	6E	10	08	00
02	00	0C	02	6E	20	02	00	21	00	D8	00	00	01	8E	00
28	D7	0E	00	01	00	00	00	00	00	00	00	01	00	00	00
01	00	00	00	01	00	00	00	05	00	00	00	01	00	00	00
09	00	0B	20	63	68	61	72	61	63	74	65	72	20	00	06
3C	69	6E	69	74	3E	00	01	43	00	0C	43	61	73	74	69
6E	67	2E	6A	61	76	61	00	01	49	00	01	4C	00	02	4C
43	00	09	4C	43	61	73	74	69	6E	67	3B	00	02	4C	49
00	02	4C	4C	00	15	4C	6A	61	76	61	2F	69	6F	2F	50
72	69	6E	74	53	74	72	65	61	6D	3B	00	12	4C	6A	61
76	61	2F	6C	61	6E	67	2F	4F	62	6A	65	63	74	3B	00
12	4C	6A	61	76	61	2F	6C	61	6E	67	2F	53	74	72	69
6E	67	3B	00	19	4C	6A	61	76	61	2F	6C	61	6E	67	2F
53	74	72	69	6E	67	42	00	6C	64	65	72	3B	00	12	00
4C	6A	61	76	61	2F	6C	61	6E	67	2F	53	79	73	74	65
6D	3B	00	01	56	00	02	56	4C	00	13	5B	4C	6A	61	76
61	2F	6C	61	6E	67	2F	53	74	72	69	6E	67	3B	00	06
61	70	70	65	6E	64	00	06	61	73	63	53	74	72	00	06
61	73	63	69	69	20	00	06	63	68	72	53	74	72	00	04
6D	61	69	6E	00	03	6F	75	74	00	07	70	72	69	6E	74
6C	6E	00	08	74	6F	53	74	72	69	6E	67	00	01	00	07
0E	00	08	01	00	07	0E	5A	01	22	0D	4D	00	02	17	14
17	00	02	00	02	00	00	18	01	18	00	81	80	04	D4	03
01	09	EC	03	0E	00	00	00	00	00	00	00	01	00	00	00
00	00	00	00	01	00	00	00	1A	00	00	00	70	00	00	00
02	00	00	00	0A	00	00	00	D8	00	00	00	03	00	00	00
07	00	00	00	00	01	00	00	04	00	00	00	03	00	00	00
54	01	00	00	05	00	00	00	09	00	00	00	6C	01	00	00
06	00	00	00	01	00	00	00	B4	01	00	00	01	20	00	00
02	00	00	00	D4	01	00	00	01	10	00	00	04	00	00	00
54	02	00	00	02	20	00	00	1A	00	00	00	72	02	00	00
03	20	00	00	02	00	00	00	7D	03	00	00	05	20	00	00
01	00	00	00	8D	03	00	00	00	20	00	00	01	00	00	00
92	03	00	00	00	10	00	00	01	00	00	00	A4	03	00	00

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

As identified above, among the organized data stored in the .dex file is a list of references specifically identifying the other modules that a given module links to. In particular, the information stored in a .dex file, which is in turn loaded or “mapped” by a portion of the Dalvik VM, includes a list of references specifically identifying the other modules that the module links to – in particular, the string\_ids, type\_ids, proto\_ids, field\_ids, method\_ids, and class\_defs sections. Accordingly, portions of the Dalvik VM comprise a data structure for storing module information.

Moreover, as explained above, portions of Dalvik VM comprise a component of the code server. Accordingly, portions of the Dalvik VM and in particular the resource pointer table(s), is a data structure of the code server that maintains module information.

b. for one or more modules of a multi-module program

The module information stored in the resource pointer table(s) is module information for one or more modules of a multi-module program. In particular, the resource pointer table(s) store module information (*i.e.*, the contents of the .dex file metadata) for classes, or modules, that are intended to be shared by multi-module programs and applications. That is, the metadata for each class loaded or “mapped” by the Dalvik VM and populated in the resource pointer table(s) is used to form a part of at least one (and typically more than one) multi-module program or application.

Accordingly, the resource pointer table(s) is a data structure that maintains module information for one or more modules of a multi-module program. And, because the identified “code server” includes the resource pointer table(s), this requirement is satisfied.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****iv. “provides to the first program upon request module information for one or more modules of a multi-module program”**

As construed by the Court in the REC/MSFT Case, the “code server” not only maintains module information for one or more modules of a multi-module program, as explained above, but it also (a) receives a request for this module information, and (b) provides this module information to the first program. The identified code server of the Accused Systems satisfies both of these additional requirements.

**a. receives a request for module information**

Computer instructions do not speak to each other in the same manner as humans speak to each other. A human requesting something from another human may say: “Can you do this for me?” But computer instructions are coded to perform specific operations, some of which require data, or information, to execute. The way that one set of computer instructions (the “requesting instructions”) requests an action from another set of computer instructions (the “receiving instructions”) is by providing information to the “receiving instructions” so that the “receiving instructions” can use that information to perform their normal encoded operation. For example, the “requesting instructions” request that the “receiving instructions” add two numbers together when the “requesting instructions” provide the two numbers to the “receiving instructions” and the normal encoded operation of the “receiving instructions” is to add two numbers together.

Accordingly, the code server receives a request to provide module information to the first program for a specific module if the code server (1) receives information that identifies the specific module, and (2) the normal encoded operation of the code server is (A) to use that information to search for module information associated with that module, and (B) to provide that module information to the “first program”. This is precisely how the code server operates in the Accused Systems:

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

*Normal encoded operation – Searches:* One of the normal encoded operations of the code server is searching for module information for a specific module, or class. This is the operation of the component of the code server which searches for a module, or class, in the resource pointer table(s). In particular, as previously described: (1) `dvmDexGetResolvedMethod(methodClassDex, ref)` causes a search of the resource pointer table(s) by Method ID: `pDvmDex->pResMethods[methodIdx]`; and (2) `dvmDexGetResolvedClass(methodClassDex, ref)` causes a search of the resource pointer table(s) by Class ID: `pDvmDex->pResClasses[classIdx]`.

*Receives identifier information:* As previously described: (1) `pDvmDex->pResMethods[methodIdx]` receives an identifier, namely `methodIdx`, that includes information about the method; and (2) `pDvmDex->pResClasses[classIdx]` receives an identifier, namely `classIdx`, that includes information about the class. In particular, the identification of the particular class and/or method that the code server receives comes (via a series of calls) from the “first program”, *i.e.*, the `dvmInterpretPortable` computer instructions of the Dalvik VM.

*Normal encoded operation – provides module information to the first program:* The normal encoded operation of the code server is not merely searching for the identified class or module in the resource pointer table(s), as detailed below in connection with the analysis of Claim 1, Element (c). It also includes providing module information to the “first program”. As detailed below in connection with the analysis of Claim 1, Element (d), function `pDvmDex->pResMethods[methodIdx]` and `pDvmDex->pResClasses[classIdx]` provides module information to the first program.

Accordingly, because (1) `dvmDexGetResolvedMethod` and `dvmDexGetResolvedClass` provide information that identifies the specific module which is received by the code server by

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

pDvmDex->pResMethods[methodIdx] and pDvmDex->pResClasses[classIdx], respectively, and (2) the normal encoded operation of the code server is (A) to use that information to search for module information associated with that module, and (B) to provide that module information to the first program, *i.e.*, dvmInterpretPortable, the code server receives a request to provide information to the “first program” for a specific module.

b. provides module information to the first program

The code server satisfies this requirement as detailed in connection with the analysis of Claim 1, Element (d), below.

Accordingly, the Accused Systems satisfy the “code server” limitation.

<b>Claim 1: Element (a)</b>
-----------------------------

“ <b><u>receiving</u></b> in a code server <b><u>from said first program</u></b> , at a point prior to execution-time of said multi-module program being associated, <b><u>a request associated with said discrete module</u></b> ”
---

This claim language is satisfied if (i) the code server receives a request that identifies the specific module and (ii) the request comes from the first program. As described above, the code server meets both of these requirements.

<b>Claim 1: Element (a)</b>
-----------------------------

“receiving in a code server from said first program, <b><u>at a point prior to execution-time of said multi-module program being associated</u></b> , a request associated with said discrete module”
---

This claim language is satisfied if the request is received by the code server at a point in time before any instruction of the multi-module program being associated is executed.

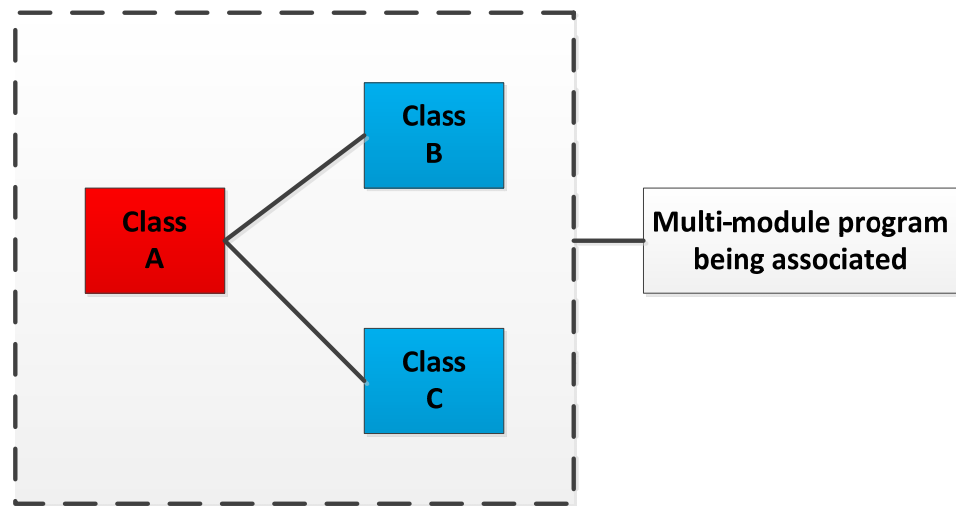
i. said multi-module program being associated

The “said multi-module program being associated” must satisfy the following two requirements: (a) First, it must be a set of computer instructions that comprises two or more modules (or classes) and enables the computer to perform a specific operation or operations; and

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

(b) Second, an association must be formed for this multi-module program pursuant to the operation claimed in Claim 1, Element (e) (*i.e.*, “forming an association of said multi-module program by said first program”).

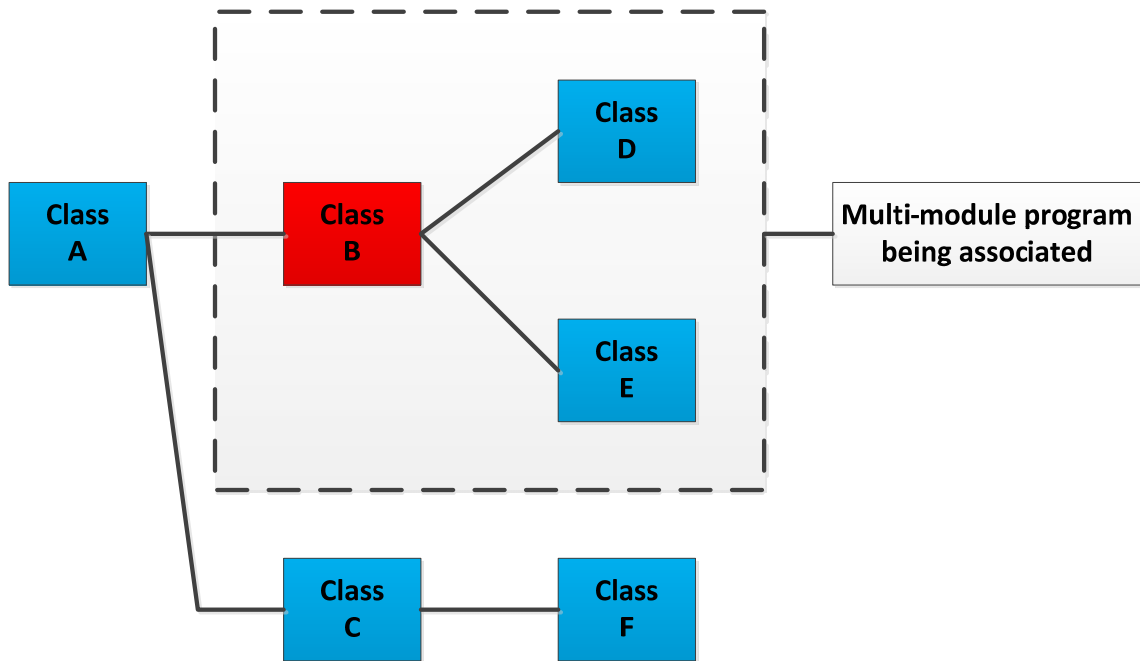
As described above, the “multi-module program being associated” consists of one class located and loaded or “mapped” from the .dex file, which is the “discrete module” referenced in the claims, and the class(es) that the class located and loaded or “mapped” from the .dex file links to. That is, with respect to a .dex program consisting of classes A-C in the manner depicted in the figure below, if class A is found in the .dex file (and is therefore the claimed “discrete module”), and class A depends upon, or links to, classes B and C, then the “multi-module program being associated” is the program consisting of classes A, B, and C.



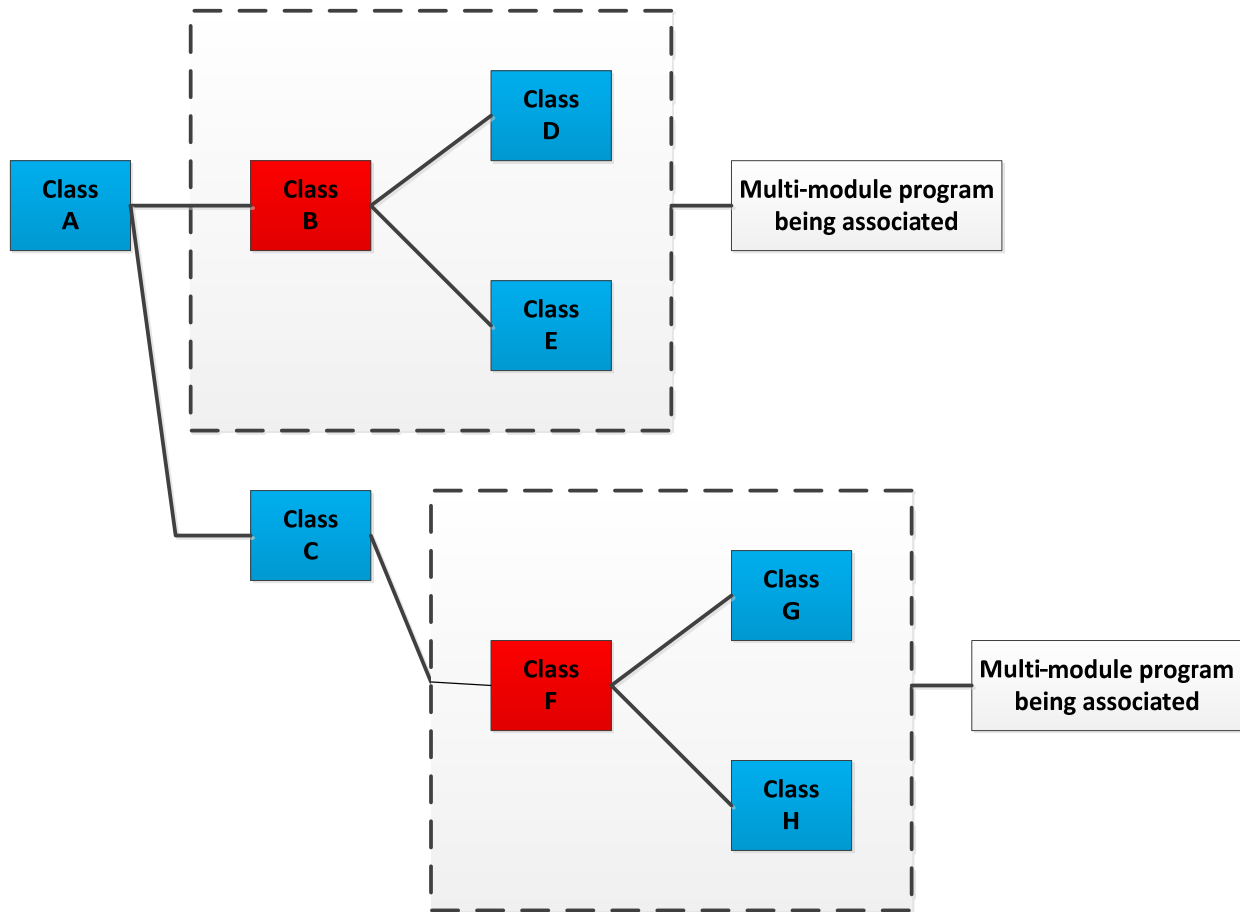
Similarly, if a .dex application consists of classes A-F in the manner depicted in the figure below, where class B is the only class of the application that necessarily resides in the .dex file, and class B links to classes D and E, then the “multi-module program being associated” is the program consisting of classes B, D, and E, *i.e.*, class B, the class found in the .dex file, is the “discrete module” referenced by the claims, and classes D and E are the classes linked to by the

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

“discrete module”. Classes A, C, and F are not part of this “multi-module program being associated”.



Any given .dex application can include several different “multi-module programs being associated”. Each such program would consist of any set of linked classes that begins with a class located in the .dex file. An example is the .dex application consisting of classes A-H in the manner depicted in the figure below, where classes B and F are stored in the .dex file. In this case, the program consisting of classes B, D, and E is a “multi-module program being associated” because class B is the claimed “discrete module” and classes D and E are the classes linked to by the “discrete module”. The program consisting of classes F, G, and H is also a “multi-module program being associated” because class F is the claimed “discrete module” and classes G and H are the classes linked to by the “discrete module”.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Moreover, every .dex application includes at least one “multi-module program being associated” because every .dex application makes use of at least one class within the .dex file, and the program consisting of that class from the .dex file and its linked to class(es) constitutes a “multi-module program being associated”.

a. First requirement

The “multi-module program being associated” must be (1) a set of computer instructions that (2) comprises two or more modules (or classes) and (3) enables the computer to perform a specific operation or operations. Each component is addressed in turn.

(1) *Set of computer instructions.* Because any given class in the .dex file is itself a set of computer instructions (*i.e.*, it consists of an executable module and corresponding metadata), the two or more classes of the identified “multi-module program being associated”

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

necessarily also constitutes a set (*i.e.*, a larger set) of computer instructions (see the analysis of “set of computer instructions” set forth above).

(2) *Comprises two or more modules (or classes).* Because the “multi-module program being associated” consists of at least two classes (*i.e.*, the class stored in the .dex file and the class or classes that the class stored in the .dex file links to), the “multi-module program being associated” comprises two or more modules (or classes).

(3) *Enables the computer to perform a specific operation or operations.* Each class in the .dex file itself enables the computer to perform a specific operation or operations. Classes are made available in the .dex file for the very reason that multi-class programs can incorporate that particular class and take advantage of the operation or operations it provides. For example, the .dex file containing the Launcher class stored in and loaded or “mapped” from the BOOTCLASSPATH and/or CLASSPATH repositories enables the computer to perform the operation of displaying the Home screen. This is useful for programs or applications that offer a return to Home screen view. Any “multi-module program being associated” that includes this particular class also enables the computer to perform the operation of displaying the Home screen.

Because a class file enables the computer to perform a specific operation, and because the identified multi-module program includes the class as a unit of the multi-module program, the multi-module program also enables the computer to perform at least the same specific operation. Accordingly, the First Requirement is satisfied.

b. Second Requirement

To be a “multi-module program being associated”, an association must be formed for the identified program consisting of one class found in the .dex file and the classes to which that

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

class found in the .dex file links to, pursuant to the operation claimed in Claim 1, Element (e) (*i.e.*, “forming an association of said multi-module program by said first program”). This requirement is satisfied as shown below in the analysis of Claim 1, Element (e).

**ii. [receiving the request] at a point in time before any instruction of the multi-module program being associated is executed**

To satisfy this claim language, the request for module information that is received by the code server must be received at a point in time before any instruction of the “multi-module program being associated” is executed. The instructions of a program are those instructions contained within the program itself. Accordingly, the instructions of a program comprised of multiple classes are the instructions included within those multiple classes.

The “multi-module program being associated” consists of the “discrete module”, which is stored and loaded or “mapped” from the .dex file, and the classes (or modules) that the discrete module links to. Accordingly, the instructions of the “multi-module program being associated” are those instructions included within the “discrete module”, which is stored and loaded or “mapped” from the .dex file, and the classes (or modules) that the discrete module links to.

Instructions of a program cannot be executed (or carried out) until at least some of the instructions of the program are located and loaded or “mapped” into memory. Accordingly, the instructions of the “multi-module program being associated” cannot be executed until at least some of the instructions of the “discrete module”, which is stored in the .dex file, or the classes (or modules) that the discrete module links to, are located and loaded or “mapped” into memory.

For the “multi-module program being associated”, the first instructions of the program that are located and loaded or “mapped” into memory are the instructions included in the discrete module located in the .dex file. The remainder of the “multi-module program” (*i.e.*, those classes, or modules, that the discrete module links to) are similarly located and loaded or

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

“mapped” into memory.

As previously described, the interpreter setup is complete at the end of `InterpC-portable.cpp` and the interpreter is ready to “fetch and execute first instruction”. Following the function `dvmInterpretPortable` in file `platform_dalvik-master\vm\mterp\out\InterpC-portable.cpp`, a list of OP CODES is included. Op codes are assembly language instructions used to execute the application Activity being run. Two op codes, `OP_CONST_CLASS` and `GOTO_TARGET`, show calls to functions that are used to locate classes and methods by causing a request for information from the resource pointer table(s) (*i.e.*, module information table), as previously described. Functions `dvmDexGetResolvedClass` and `dvmDexGetResolvedMethod` are found in file `platform_dalvik-master\vm\DvmDex.h`. Function `dvmDexGetResolvedClass` causes a search of the resource pointer table(s) by Class ID: `pDvmDex->pResClasses[classIdx]`. Function `dvmDexGetResolvedMethod` causes a search of the resource pointer table(s) by Method ID: `pDvmDex->pResMethods[methodIdx]`. The data resulting from the search of the resource pointer table(s) is then used to locate the linked to class file, which is then subsequently executed.

Accordingly, locating and loading or “mapping” into memory the “discrete module” occurs at a point prior to executing any instruction of the “multi-module program being associated”.

The “discrete module” is not located in mapped memory until a point in time after the code server receives the request associated with the discrete module. The purpose, and the subsequent result of, receiving the request associated with the discrete module is searching for and providing module information relating to the discrete module so that its contents can be

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

executed. That is, the locating of the mapped discrete module in memory occurs at a point in time after receiving the request for the discrete module.

Accordingly, because (a) the request associated with the discrete module is received at a point in time prior to locating the mapped discrete module in memory, and (b) locating the mapped discrete module in memory occurs at a point in time prior to executing any instruction of the “multi-module program being associated”, it necessarily follows that the request for the discrete module is received at a point in time prior to executing any instruction of the “multi-module program being associated”.

<b>Claim 1, Element (a): Conclusion</b>
---

The Accused Systems satisfy this element.

<b>Claim 1, Element (a): Doctrine of Equivalents</b>
--

In the alternative, the Accused Systems meet this element under the doctrine of equivalents because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.

<b>Claim 1</b>
<b>Element (b): Claim Language</b>
“searching a module information table for module information in response to said request associated with said discrete module”

This element can be broken down into two parts: (i) a “module information table” that stores “module information;” and (ii) “searching a module information table for module information in response to said request associated with said discrete module.”

**i. a “module information table” that stores “module information”**

Applying the agreed constructions, an Accused System includes a “module information table” that stores “module information” if it includes “a data structure of the code server (a) for

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

storing module information (b) for multiple modules (c) independent of both the executable modules themselves and the first program.”

The Accused Systems satisfy this claim language:

a. a data structure of the code server for storing module information

A data structure for storing module information is an organized collection of data that stores information about a module that includes at least a list of references specifically identifying the other modules that the module links to.<sup>51</sup> The resource pointer table(s) of the Accused Systems is (i) a component of the code server and (ii) an organized collection of data that stores information about a module that includes at least a list of references specifically identifying the other modules that the module links to.

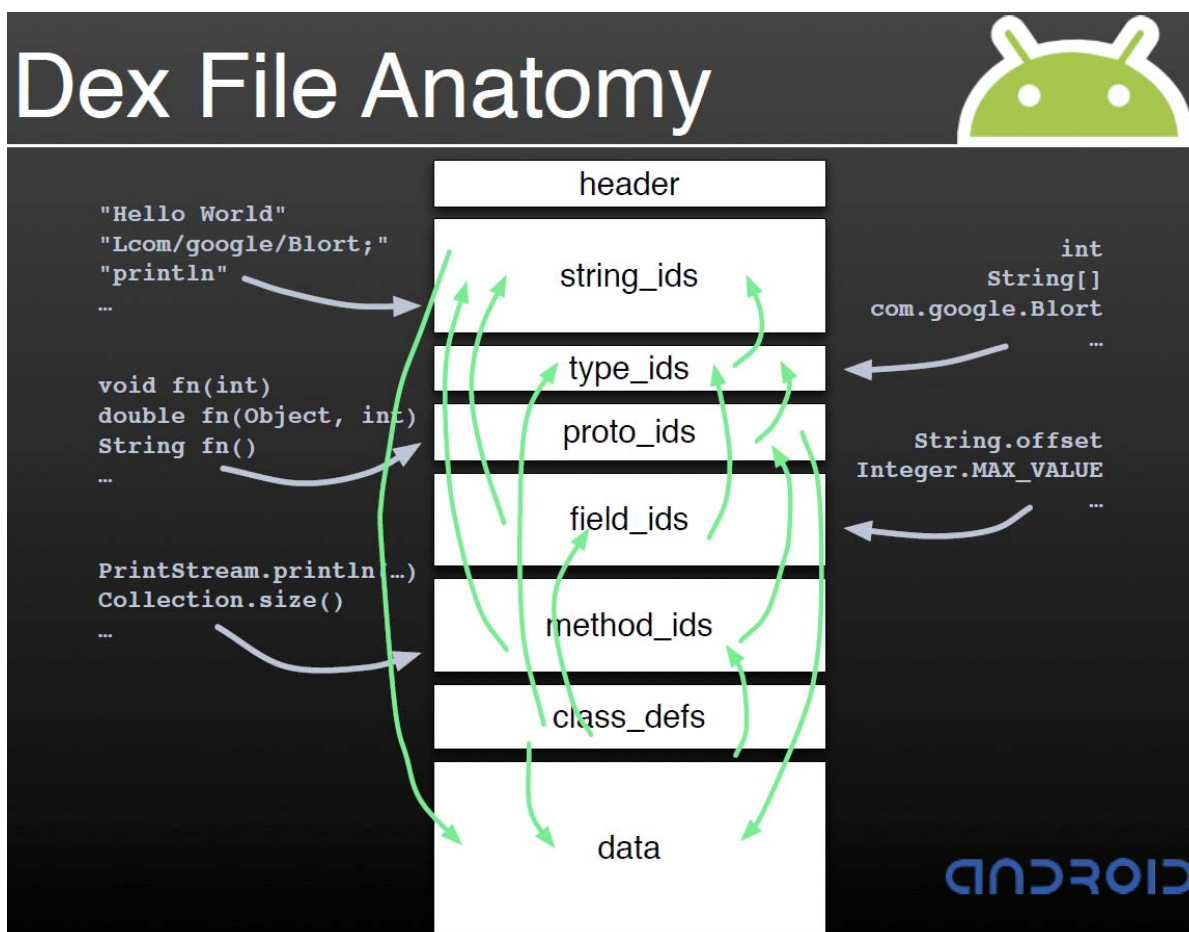
In particular, the resource pointer table(s) stores module information in an organized manner. The image below<sup>52</sup> illustrates the .dex file maintaining module information in an organized manner:

---

<sup>51</sup> See, e.g., *McGraw-Hill Dictionary of Electronics and Computer Technology* (1984) (defining “data structure” as “a collection of data components that are constructed in a regular and characteristic way.”); *Wiley Electrical and Electronics Engineering Dictionary* (2004) (defining “data structure” as “an organizational scheme, such as a record or array, that can be applied to data to facilitate interpreting the data of performing operations on it.”); *Microsoft Computer Dictionary* (4th ed. 1999) (defining “data structure” as “an organizational scheme, such as a record or array, that can be applied to data to facilitate interpreting the data or performing operations on it”).

See also Agreed Terms, construing “module information” as “information about a module that includes at least a list of references specifically identifying the other modules that the module links to”.

<sup>52</sup> Dan Bornstein, “Dalvik VM Internals”, Presented at Google I/O Session, 2008, <https://sites.google.com/site/io/dalvik-vm-internals>; <https://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The .dex file consists of data, including not only the executable code of the module (*i.e.*, class file), but also organized metadata corresponding to that module.

Metadata corresponding to a class in the .dex file includes information about the module stored in the .dex file. This information includes, for example, class name strings, method name strings, method “signature” (prototype), and pointers to code location in memory.

As identified above, among the organized data stored in the .dex file is a list of references specifically identifying the other modules that a given module links to. In particular, the information stored in the .dex file, that when loaded or “mapped” is in turn populated in the resource pointer table(s) in an organized manner, includes a list of references specifically identifying the other modules that the module links to – in particular, the names of these linked

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

to modules, or dependencies. Accordingly, the resource pointer table(s) is a data structure for storing module information. Moreover, as demonstrated in connection with element (a), the resource pointer table(s) is a component of the code server.

Accordingly, the resource pointer table(s) is a data structure of the code server for storing module information.

b. module information for multiple modules

The module information populated in the resource pointer table(s) is not for one particular module, or class. That is, the resource pointer table(s) does not store just one set of module information corresponding to one module. Instead, the resource pointer table(s) stores a set of module information for a plurality of modules – in particular, all modules, or classes, that are used by multiple classes of the .dex file.

c. module information independent of the executable modules themselves and the first program independent of the executable modules

There is a distinction between traditional modules, which may include references within the executable code to other modules that they depend upon, and .dex files, which include an additional, independent logical component, *i.e.*, the metadata, which exists as a source of information about the module.

The '936 patent discusses, as Background, the traditional prior art systems like an OS/2 operating system, which included linkage information co-mingled within the executable modules themselves. To determine which other modules a given module links to, OS/2 would scan the contents of the executable modules and “extract the needed code module information” from the code modules, which was an “inefficient” process.<sup>53</sup>

---

<sup>53</sup> '936 patent, 1:31-32.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

In contrast, the Accused Systems have added an additional component – the metadata – that is logically independent of the executable code. The additional component (*i.e.*, the metadata table) can be stored in two ways – (1) in the same file with the executable module, where the metadata is in addition, logically independent component of the executable module, or (2) as an additional file, where the metadata is both a logically and physically independent component of the class.

Moreover, by storing the module information in metadata independent from the modules themselves, the Accused Systems experience the advantage of avoiding the inefficient process of scanning and extracting linkage information from within the executable code. Instead, when running a program, the Accused Systems determine the needed linked-to modules to load or “map” by consulting the dependencies in the metadata, not by extracting linkage information from within the executable code.

The module information stored and loaded or “mapped” from the .dex file also stores module information independent of the first program. We know this is true for several reasons.

*First*, the first program (*i.e.*, portions of the Dalvik Virtual Machine) is not stored in the .dex file from which the metadata is obtained for the module information table of the code server. The .dex file is stored in the BOOTCLASSPATH and/or CLASSPATH repositories. Because the first program is not even stored in the BOOTCLASSPATH and/or CLASSPATH repositories, the module information obtained from the metadata in .dex files that are stored in the BOOTCLASSPATH and CLASSPATH repositories is necessarily stored independently from the first program.

*Second*, as explained above, the first program plays a role in causing the search for the module information that was obtained from the .dex file in the BOOTCLASSPATH and/or

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

CLASSPATH repositories for loading/”mapping” and execution. If the first program were itself stored in the BOOTCLASSPATH and/or CLASSPATH repositories, this would result in the situation where certain code was initiating a search for itself. This is logically impossible in the Android environment. The code cannot initiate a search for itself, because that code has not yet been found.

*Third*, even if the first program were stored in the BOOTCLASSPATH and/or CLASSPATH repositories, this claim element is still satisfied. As detailed above, because module information obtained from each .dex file in the BOOTCLASSPATH and/or CLASSPATH repositories is stored independently from every other .dex file in the BOOTCLASSPATH and/or CLASSPATH repositories. Accordingly, even if the first program were stored in the BOOTCLASSPATH and CLASSPATH repositories, this module information obtained from the .dex file would still be stored independently from all other .dex files in the BOOTCLASSPATH and/or CLASSPATH repositories. Accordingly, the module information for each of the non-first program classes would be stored independently from the module information for the first program classes.

**ii. searching a module information table for module information in response to said request associated with said discrete module”**

The code server of the Accused Systems satisfies this claim language, as construed by the Court in the REC/MSFT Case, if the code server (a) searches for module information in the module information table pertaining to the discrete module (b) in response to the request that identifies the discrete module.

As discussed in the analysis of the previous element, the resource pointer table(s) is the “module information table,” and the information (including linkage information) loaded or “mapped” from the metadata of a .dex file is the “module information”. Accordingly, this

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

element is satisfied if, for the “discrete module” (*i.e.*, the module or class referenced in the previous element), the code server searches for the corresponding metadata in the resource pointer table(s), in response to a request that identifies the discrete module.

As demonstrated above, the functions `dvmDexGetResolvedClass` and `dvmDexGetResolvedMethod` are found in file `platform_dalvik-master\vm\DvmDex.h`. Function `dvmDexGetResolvedClass` causes the code server to search its Resource pointer table(s) by Class ID: `pDvmDex->pResClasses[classIdx]`. Function `dvmDexGetResolvedMethod` causes the code server to search its Resource pointer table(s) by Method ID: `pDvmDex->pResMethods[methodIdx]`. The `dvmDexGetResolvedMethod(methodClassDex, ref)` and `dvmDexGetResolvedClass(methodClassDex, ref)` functions include a request that identifies the class or method, respectively, by name.

For example, if a program consists of three classes – Class A, which links to Classes B and C – in response to a request to search for Class A, the code server will search for the module information associated with Class A. However, because the search for Class A resulted in a need for Classes B and C (*i.e.*, the dependency information of Class A indicated that Classes B and C are needed), the searching process will need to be repeated for Classes B and C. That is, as a result of the request for Class A (which resulted in the need for Classes B and C), additional searches will need to be run for module information for Classes B and C. The first place these classes will be searched for is the resource pointer table(s).

It is possible that module information Classes B and C may not be present in the resource pointer table(s) in which case the code server will search elsewhere for the module information. But, in any event, if these additional classes are needed (*i.e.*, they are part of the program being

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

executed), the code server will always search for these referenced modules in the resource pointer table, and this additional limitation is satisfied.

<b>Claim 1, Element (b): Conclusion</b>
---

The Accused Systems satisfy this element.

<b>Claim 1, Element (b): Doctrine of Equivalents</b>
--

In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.

<b>Claim 1</b>
<b>Element (c): Claim Language</b>
“in response to finding said module information in said module information table, reading said module information”

This claim element is satisfied if, in response to finding “module information” pertaining to the particular module in the “module information table”, the Accused System reads the module information.

As previously discussed, the resource pointer table(s) is the “module information table”, and the information (including linkage information) loaded or “mapped” from the metadata of a .dex file is the “module information”. Accordingly, this element is satisfied if, for the “discrete module” (*i.e.*, the module or class referenced in the previous element), the code server locates the corresponding metadata in the resource pointer table(s) as identified in the request for the discrete module from the first program; and reads back a reference to that metadata in the resource pointer table(s) in its response to the first program.

As explained in Application Execution, above, the first program, `dvmInterpretPortable` (the interpreter), uses function `dvmDexGetResolvedClass(methodClassDex, ref)` or

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

dvmDexGetResolvedMethod(methodClassDex, ref) to make a request from the code server (Claim 1, Element (a)). The code server searches the module information table (Claim 1, Element (b)) using pDvmDex->pResClasses[classIdx] or pDvmDex->pResMethods[methodIdx], respectively, as shown in the following code snippet:

```
platform_dalvik-master\vm\DvmDex.h

INLINE struct ClassObject* dvmDexGetResolvedClass(constDvmDex* pDvmDex,
u4 classIdx)
{
    assert(classIdx < pDvmDex->pHeader->typeldsSize);
    return pDvmDex->pResClasses[classIdx];
}

INLINE struct Method* dvmDexGetResolvedMethod(constDvmDex* pDvmDex,
u4 methodIdx)
{
    assert(methodIdx < pDvmDex->pHeader->methodIdsSize);
    return pDvmDex->pResMethods[methodIdx];
}
```

In the above code, the code server reads data found at class index classIdx or method index methodIdx, respectively, in the module information table and returns a reference to that data as a response (Claim 1, Element (d)) to the first program's request (Claim 1, Element (a)).

**Claim 1, Element (c): Conclusion**

The Accused Systems satisfy this element.

**Claim 1, Element (c): Doctrine of Equivalents**

In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

<b>Claim 1</b>
<b>Element (d): Claim Language</b>
“providing said module information in a response to said first program”

The Accused Systems satisfy this element if they include computer instructions that provide the “module information” that was requested and searched for in a response to the “first program”, as described in connection with the analysis of Claim 1, Element (b), above.

That “module information” is information populated in the resource pointer table(s) for the “discrete module” (*i.e.*, a portion of the metadata obtained from the .dex file for the class of the “multi-module program” located in the BOOTCLASSPATH and/or CLASSPATH repositories), as described in the analysis of Claim 1, Element (b), above. The “first program” is portions of the Dalvik VM, which consists of the computer instructions responsible for the operation of forming a data structure, as described in the analysis of Claim 1, Element (e), below.

Accordingly, this element is satisfied if (i) the Accused Systems include computer instructions that provide contents of the resource pointer table(s) for the “discrete module” (*i.e.*, the class of the multi-module program stored in the .dex file) to `dvmGetResolvedMethod` and/or `dvmGetResolvedClass`, and (ii) contents of the resource pointer table(s) are provided in a “response”. These two requirements are addressed in turn.

**i. provide contents of the resource pointer table(s) to `dvmGetResolvedMethod` and/or `dvmGetResolvedClass`**

To “provide” means “to make available or furnish”.<sup>54</sup> The term “provide” has no special meaning to one of ordinary skill in the art. Accordingly, in the context of computer instructions, one set of computer instructions (the “providing instructions”) provides information to another set of computer instructions (the “receiving instructions”) by making information available to the

---

<sup>54</sup> See Dictionary.com, “provide”, <http://dictionary.reference.com/browse/provide>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

receiving instructions. This can be accomplished, for example, by (a) the “providing instructions” creating a data structure to store the information, when (b) the normal encoded operations of the “receiving instructions” is to make use of that data structure.

Accordingly, this sub-requirement is satisfied if (a) the Accused Systems includes instructions that create a data structure that stores the contents of the metadata of the “discrete module”, *i.e.*, the module information that was obtained for the class of the “multi-module program” stored in the .dex file, and (b) the normal encoded operation of `dvmDexGetResolvedMethod` and `dvmDexGetResolvedClass` is to make use of this data structure. Both components of this requirement are satisfied.

The `dvmDexGetResolvedMethod` and `dvmDexGetResolvedClass` components of the “first program” of the Accused Systems loads or “maps” the contents of the metadata of the “discrete module” into the `dvmInterpPortable` program, as described in the analysis of Claim 1, Element (c), above.

The normal encoded operation of `dvmInterpPortable` makes use of the data structure provided by the code server. In particular, the normal encoded operation of `dvmInterpPortable` is to create a data structure and to populate that data structure with the metadata of a given program.

That is, the `dvmDexGetResolvedMethod` and `dvmDexGetResolvedClass` components of the first program result in module information being provided to `dvmInterpPortable`. Accordingly, because (a) the code server of the Accused Systems creates a data structure that stores the contents of the metadata of the “discrete module”, and (b) the normal encoded operation of `dvmInterpPortable` is to make use of this data structure created by the code server,

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

the code server provides contents of the metadata of the class in the resource pointer table(s) to `dvmInterepPortable`.

Moreover, because the `allocateAuxStructures` component of the code server provides contents of the metadata of the `.dex` file from the `BOOTCLASSPATH` and `CLASSPATH` repositories to the `dvmInterepPortable` (by way of providing a data structure that includes these contents), the contents of the `dvmInterepPortable` data structure now also include the contents of the metadata from the `.dex` file.

**ii. the contents of the resource pointer table(s) are provided in a “response”**

The contents of the resource pointer table(s) provided to `dvmDexGetResolvedMethod` and `dvmDexGetResolvedClass` must be provided in a response. For there to be a response, there must first be a request. A response provides the information sought by the request.

As demonstrated above, the code server receives a request to provide “module information” (*i.e.*, contents of the metadata) for a specific module (*i.e.*, the “discrete module”) to `dvmInterepPortable` (*i.e.*, the “first program”). As further demonstrated above, as a result of this request, the code server provides the requested information (*i.e.*, contents of the metadata of the discrete module) back to the first program in the response from `dvmDexGetResolvedMethod` and `dvmDexGetResolvedClass`.

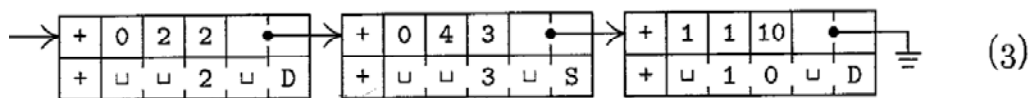
The resource pointer table(s) may be provided in the form of pointers, which is equivalent to a table.

In computer programming, information does not need to physically reside in data structure to be deemed included in that data structure. One of the most common ways to include information in a data structure is by including within that data structure pointers to the information residing in other data-structures. The definitive reference on data structures is D. E.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

Knuth, "The Art of Computer Programming", vol. 1, (Addison Wesley Longman), first published in the 1960s and most recently revised in 1997. Chapter 2 of that volume is Information Structures, and most of the chapter concerns how to construct and use various sorts of data structures using pointers, which Knuth prefers to call links (not to be confused with dynamic links, or linkage information, that is discussed throughout this report). Knuth uses the terms "link" and "pointer" interchangeably. In computer science, they mean the same thing:

The introduction of links [i.e., pointers] to other elements is an extremely important idea in computer programming; links are the key to the representation of complex structures. ... Notice also that (3) indicates the top card [in a data structure representing a group of playing cards] by an arrow from "TOP"; here TOP is a link variable, often called a pointer variable, namely a variable whose value is a link. All references to nodes in a program are made directly through link variables.



It is universally accepted to those skilled in the art that "links" in a given data structure, and the contents of what is linked to, are themselves contents of that data structure. Were it otherwise, complex data structures would be unwieldy and impractical. For example, a standard data structure is a "linked list", in which each item in a sequence is stored in a small data structure with a link (pointer) to the next item. By using the links, it is easy to step through the list in sequence. Imagine a linked list containing items A, B, D, E, F, G, and a programmer needs to insert item C between B and D. If the list were stored sequentially in a single storage area, first the program would have to move items D, E, F, and G in order to make room for C. But using a linked list, item C can be placed in any available storage location, and the links adjusted so that C logically is between B and D. In applications that manage long lists of data, these techniques

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

save a great deal of time that would otherwise be spend shuffling list items back and forth to create space for new items and close up space from deleted items.

Linked lists are also an essential technique to save space and avoid data duplication. Imagine an application that maintains a list of items to be delivered, and the addresses to deliver them to. If each address is stored along with the item, in the common case that several items are going to the same address, the address information is duplicated for each item. If the addresses are stored separately, with a link from the item to the address, all the items for a given address can link to the same copy of the address, saving a great deal of storage. Now imagine that one of the recipients has moved, so her address needs to be updated. In the linked scenario, a single update to her address suffices to update all of the packages to be delivered, while if the addresses were stored with each item, they'd have to be updated in each place they occur, taking extra time and introducing a potential source of inconsistent data if the addresses aren't all updated at the same time.

These and other list and pointer management techniques are taught in every undergraduate computer science curriculum, and are well known to any programmer of ordinary skilled in the art.

<b>Claim 1, Element (d): Conclusion</b>
---

The Accused Systems satisfy this element.

<b>Claim 1, Element (d): Doctrine of Equivalents</b>
--

In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

<b>Claim 1</b>
<b>Element (e): Claim Language</b>
“forming an association of said multi-module program by said first program”

This element is satisfied under the Court’s claim construction if (i) there is a first program; (ii) the first program forms a data structure; and (iii) the data structure includes information concerning how one or more modules of the second multi-module program links to one or more other modules. The Accused Systems satisfy each of these requirements.

**i. the first program**

The identified “first program” includes `dvmInterpretPortable`. The terms “first” and “program” are addressed in turn.

**a. first**

The term “first” merely distinguishes the “first program” from “second multi-module program”, *i.e.*, the first and second program are two different programs. As detailed throughout these infringement contentions, the “second multi-module program” is the program that the operating system is attempting to load or “map” and run, whereas the `dvmInterpretPortable` program is part of the Dalvik VM itself that is working to load or “map” and run the second multi-module program. Accordingly, the “first program” and “second multi-module program” are different programs.

**b. program**

As explained above, a “program” is a “set of computer instructions that enables the computer to perform a specific operation or operations”. `dvmInterpretPortable` comprises the computer instructions responsible for fetching and executing computer instructions found in the multi-module program, thereby forming a data structure that points to linkage information

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

between the calling module and the called module for each class of a given program, or application, being executed.

As explained in Application Execution, above, the first program, `dvmInterpretPortable` (the interpreter), uses function `dvmDexGetResolvedClass(methodClassDex, ref)` or `dvmDexGetResolvedMethod(methodClassDex, ref)` causing a request for information from the code server (Claim 1, Element (a)). The code server searches the resource pointer table(s) by Class ID: `pDvmDex->pResClasses[classIdx]` and the resource pointer table(s) by Method ID: `pDvmDex->pResMethods[methodIdx]` (Claim 1, Element (b)). The code server searches the Module Information Table (Claim 1, Element (b)) and locates and reads or “maps” module information that the first program uses to form an association (Claim 1, Element (c)). The code server provides the module information as a response (Claim 1, Element (d)) to the first program’s request (Claim 1, Element (a)).

The following code snippet shows requests in the form of `dvmDexGetResolvedClass` and `dvmDexGetResolvedMethod`, responses in the form of **return** `pDvmDex->pResClasses[classIdx]` and **return** `pDvmDex->pResMethods[methodIdx]`, and the data structures formed by the first program, **struct** `ClassObject*` and **struct** `Method*`:

```
platform_dalvik-master\vm\DvmDex.h
```

```

INLINE struct ClassObject* dvmDexGetResolvedClass(constDvmDex* pDvmDex,
u4 classIdx)
{
    assert(classIdx < pDvmDex->pHeader->typeldsSize);
    return pDvmDex->pResClasses[classIdx];
}

```

```

INLINE struct Method* dvmDexGetResolvedMethod(constDvmDex* pDvmDex,
u4 methodIdx)
{
    assert(methodIdx < pDvmDex->pHeader->methodldsSize);
    return pDvmDex->pResMethods[methodIdx];
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****ii. The first program forms a data structure**

A data structure is an organized collection of data that stores information.<sup>55</sup> The “first program”, `dvmInterpretPortable`, forms a collection of data that stores information that includes mapped information from the metadata of classes from the `.dex` file. This information is not organized in a haphazard fashion, where it would be of little use. Instead it is stored in a data structure, as shown in the following code snippet:

```
platform_dalvik-master\vm\DvmDex.h
/*
 * Some additional VM data structures that are associated with the DEX file.
 */
structDvmDex {
/* pointer to the DexFile we're associated with */
DexFile*      pDexFile;

/* clone of pDexFile->pHeader (it's used frequently enough) */
constDexHeader* pHeader;

/* interned strings; parallel to "stringIds" */
structStringObject** pResStrings;

/* resolved classes; parallel to "typeIds" */
structClassObject** pResClasses;

/* resolved methods; parallel to "methodIds" */
structMethod** pResMethods;

/* resolved instance fields; parallel to "fieldIds" */
/* (this holds both InstField and StaticField) */
structField** pResFields;

/* interface method lookup cache */
structAtomicCache* pInterfaceCache;
```

---

<sup>55</sup> See, e.g., *McGraw-Hill Dictionary of Electronics and Computer Technology* (1984) (defining “data structure” as “a collection of data components that are constructed in a regular and characteristic way.”); *Wiley Electrical and Electronics Engineering Dictionary* (2004) (defining “data structure” as “an organizational scheme, such as a record or array, that can be applied to data to facilitate interpreting the data of performing operations on it.”); *Microsoft Computer Dictionary* (4th ed. 1999) (defining “data structure” as “an organizational scheme, such as a record or array, that can be applied to data to facilitate interpreting the data or performing operations on it”).

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

/* shared memory region with file contents */
bool isMappedReadOnly;
MemMapping memMap;

jobject dex_object;

/* lock ensuring mutual exclusion during updates */
pthread_mutex_t modLock;
};

```

**iii. The DvmDex data structures, including pResClasses (a type ClassObject<sup>xxxviii</sup> data structure) and pResMethods (a type Method<sup>xxxix</sup> data structure) include information concerning how one or more modules of the second multi-module program link to one or more other modules**

As explained in the analysis of Claim 1, Element (e), set forth above, the data structures formed by `dvmInterpretPortable` include `pResClass` and `pResMethods`. Further, the code snippet set forth above in section (ii) of the analysis of Claim (1), Element (e), includes the comments “resolved classes; parallel to ‘typeIds’” and “resolved methods; parallel to ‘methodIds’”:

```

platform_dalvik-master\vm\DvmDex.h
/* resolved classes; parallel to "typeIds" */
struct ClassObject** pResClasses;

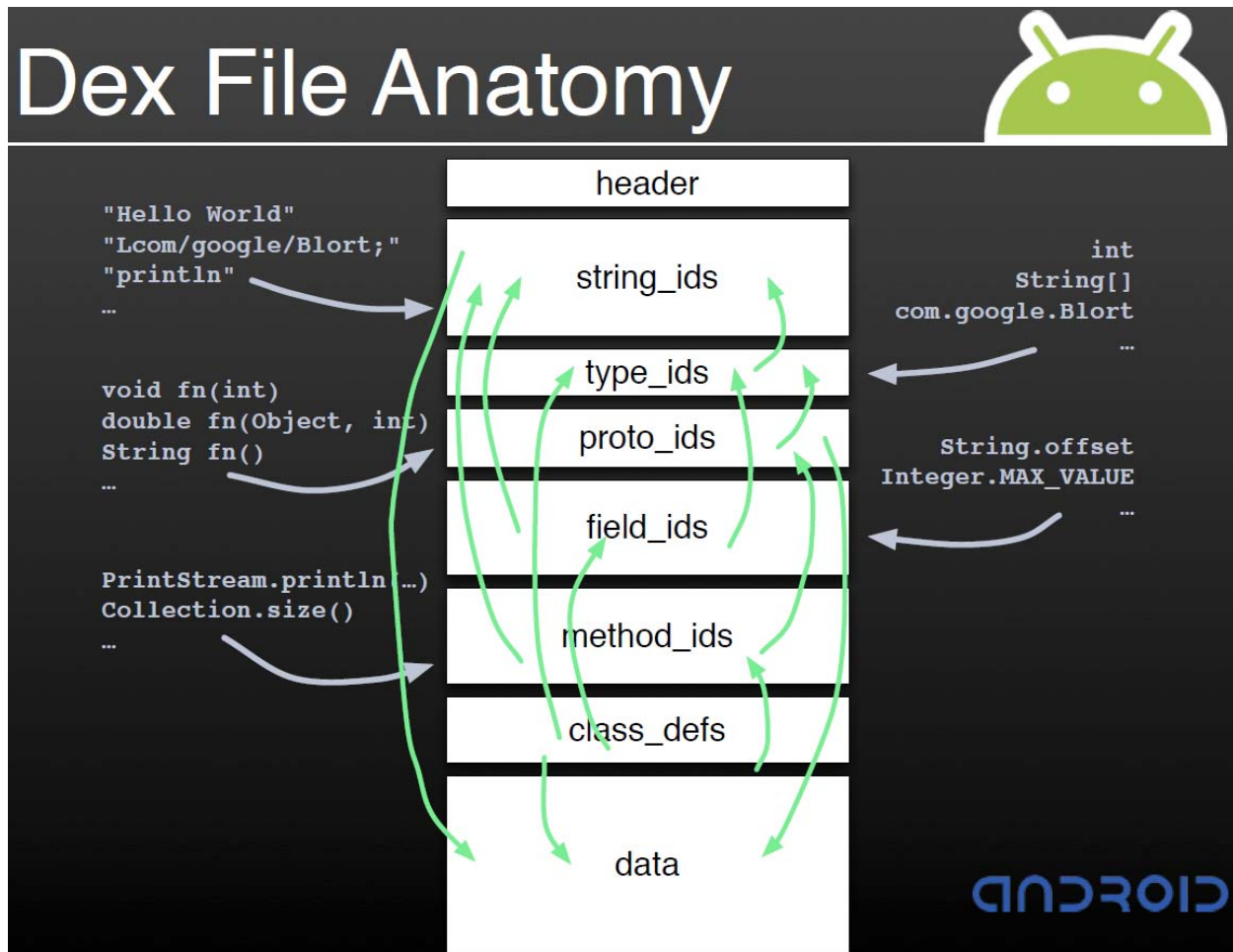
/* resolved methods; parallel to "methodIds" */
struct Method** pResMethods;

```

These comments mean that the `pResClasses` data structure includes the same linkage information as (is “parallel to”) “typeIds”; and that the `pResMethods` data structure includes the same linkage information as (is “parallel to”) “methodIds”, which refers to the previously described mapped information from the metadata of classes from the `.dex` file. In Dex Metadata, above, explained how `.dex` file metadata structures “typeIds” and “methodIds” link together one or more modules of the second multi-module program to one or more other modules.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

The figure below<sup>56</sup> illustrates the linkage pointers found in the various .dex file metadata sections:



As previously explained, the “second multi-module program” is a .dex program consisting of (a) one class found in the .dex file, which is the “discrete module” referenced in the claims, and (b) any classes that the discrete module links to.

As also previously explained, any class stored in the .dex file includes metadata that includes information concerning how that particular class (or module) links to other classes (or modules). The contents of the metadata of a .dex file that includes one or more classes are

<sup>56</sup> Dan Bornstein, “Dalvik VM Internals”, Presented at Google I/O Session, 2008, <https://sites.google.com/site/io/dalvik-vm-internals>; <https://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0>.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

loaded or “mapped” into a resource pointer table(s) data structure. Accordingly a resource pointer table(s) includes information concerning how that particular class (or module) links to other classes (or modules).

Because (a) the resource pointer table(s) structure includes information concerning how one or more modules of the second multi-module program link to one or more other modules; and (b) the first program, `dvmInterpretPortable`, uses functions `dvmDexGetResolvedClass(methodClassDex, ref)` or `dvmDexGetResolvedMethod(methodClassDex, ref)` to make requests from the code server and the code server queries the module information table to extract the linkage information and (c) the first program organizes the linkage information into `pResClasses` and `pResMethods` data structures, this requirement is satisfied.

Accordingly, the Accused Systems satisfy all three requirements under the applicable construction.

<b>Claim 1, Element (e): Conclusion</b>
---

The Accused Systems satisfy this element.

<b>Claim 1, Element (e): Doctrine of Equivalents</b>
--

In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

<b>Claim 2</b>	
<b>Claim Language</b>	<b>Analysis</b>
<b>(Preamble) “The method of claim 1 further comprising the step of,”</b>	<i>See</i> Claim 1 analysis.
<b>“in response to not finding said module information in said module information table, searching for said discrete module in a storage area.”</b>	<p>The “said module information” is identified in Claim 1, Element (b).</p> <p>The “said module information table” is identified in Claim 1, Element (b).</p> <p>The “said discrete module” is identified in Claim 1, Preamble.</p> <p>The “storage area” comprises “CLASSPATH” and/or “BOOTCLASSPATH” where .dex files containing discrete modules (classes) are found. <i>See, e.g.</i>, file</p> <pre>platform_dalvik-master\vm\oo\Class.h : /*  * The classpath and bootclasspath differ in that only the latter is  * consulted when looking for classes needed by the VM. When searching  * for an arbitrary class definition, we start with the bootclasspath,  * look for optional packages (a/k/a standard extensions), and then try  * the classpath.  *  * In Dalvik, a class can be found in one of two ways:  * - in a .dex file  * - in a .dex file named specifically "classes.dex", which is held  * inside a jar file  *  * These two may be freely intermixed in a classpath specification.  * Ordering is significant.  */</pre> <p>The “searching for said discrete module in a storage area” occurs when one or more portions of the Dalvik VM look through, examine, or otherwise search “a storage area” (identified above) for the “discrete module” (identified above).</p>

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

	<p>This “searching” occurs “in response to not finding said module information in said module information table” – <i>i.e.</i>, after, or as a result of, the “searching” process (identified in Claim 1, Element (b)) not arriving at, or not locating, certain “module information” (identified above) in the “module information table” (identified above).</p> <p>If the associated module is not found in the “module information table,” then function <code>dvmResolveClass(curMethod-&gt;clazz, ref, true)</code> in file <code>platform_dalvik-master\vm\oo\Resolve.cpp</code> is called:</p> <pre> /*  * Find the class corresponding to "classIdx", which maps to  * a class name  * string. It might be in the same DEX file as "referrer", in a  * different  * DEX file, generated by a class loader, or generated by the  * VM (e.g.  * array classes).  *  * Because the DexTypeld is associated with the referring  * class' DEX file,  * we may have to resolve the same class more than once if  * it's referred  * to from classes in multiple DEX files. This is a necessary  * property for  * DEX files associated with different class loaders.  *  * We cache a copy of the lookup in the DexFile's "resolved  * class" table,  * so future references to "classIdx" are faster.  *  * Note that "referrer" may be in the process of being linked.  *  * Traditional VMs might do access checks here, but in  * Dalvik the class  * "constant pool" is shared between all classes in the DEX  * file. We rely  * on the verifier to do the checks for us.  *  * Does not initialize the class.  *  * "fromUnverifiedConstant" should only be set if this call is  * the direct  * result of executing a "const-class" or "instance-of"  * instruction, which  * use class constants not resolved by the bytecode verifier.  *  * Returns NULL with an exception raised on failure. </pre>
--	--

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

*/
ClassObject* dvmResolveClass(const ClassObject*
referrer, u4 classIdx,
    bool fromUnverifiedConstant)
{
    DvmDex* pDvmDex = referrer->pDvmDex;
    ClassObject* resClass;
    const char* className;

    /*
     * Check the table first -- this gets called from the other
     "resolve"
     * methods.
     */
    resClass = dvmDexGetResolvedClass(pDvmDex,
classIdx);
    if (resClass != NULL)
        return resClass;

    LOGVV("--- resolving class %u (referrer=%s cl=%p)",
        classIdx, referrer->descriptor, referrer->classLoader);

    /*
     * Class hasn't been loaded yet, or is in the process of
     being loaded
     * and initialized now. Try to get a copy. If we find one,
     put the
     * pointer in the DexTypeIdx. There isn't a race condition
     here --
     * 32-bit writes are guaranteed atomic on all target
     platforms. Worst
     * case we have two threads storing the same value.
     *
     * If this is an array class, we'll generate it here.
     */
    className = dexStringByTypeIdx(pDvmDex->pDexFile,
classIdx);
    if (className[0] != '\0' && className[1] == '\0') {
        /* primitive type */
        resClass = dvmFindPrimitiveClass(className[0]);
    } else {
        resClass = dvmFindClassNoInit(className, referrer-
>classLoader);
    }
}

```

In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

<b>Claim 3</b>	
<b>Claim Language</b>	<b>Analysis</b>
<b>(Preamble) “The method of claim 2 further comprising the step of,”</b>	<i>See</i> Claim 2 analysis.
<b>“in response to finding said discrete module in said storage area, updating said module information table with said module information relating to said discrete module.”</b>	<p>The “said discrete module” is identified in Claim 1, Preamble.</p> <p>The “said storage area” is identified in Claim 2.</p> <p>The “said module information table” is identified in Claim 1, Element (b).</p> <p>The “said module information” is identified in Claim 1, Element (b).</p> <p>The “updating said module information table with said module information relating to said discrete module” occurs after the discrete module is found in storage. In file platform_dalvik-master\vm\oo\Resolve.cpp, function dvmDexSetResolvedClass is called to update the module information table:</p> <pre> /*  * Add what we found to the list so we can skip the class search  * next time through.  *  */ dvmDexSetResolvedClass(pDvmDex, classIdx, resClass); </pre> <p>And in file platform_dalvik-master\vm\DvmDex.h:</p> <pre> /*  * Update the resolved item table. Resolution always produces the same  * result, so we're not worried about atomicity here.  */ INLINE void dvmDexSetResolvedClass(DvmDex* pDvmDex, u4 classIdx, struct ClassObject* clazz) { assert(classIdx &lt; pDvmDex-&gt;pHeader-&gt;typeldsSize); pDvmDex-&gt;pResClasses[classIdx] = clazz; } </pre>

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

	<p>The “updating” occurs “in response to finding said discrete module in said storage area” – <i>i.e.</i>, after, or as a result of, the “searching for said discrete module in a storage area” (identified in Claim 2 above) arriving at, or locating, the “discrete module” (identified above).</p> <p>In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.</p>
--	--

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

<b>Claim 4</b>	
<b>Claim Language</b>	<b>Analysis</b>
<b>(Preamble) “The method of claim 2 further comprising the step of,”</b>	<i>See</i> Claim 2 analysis.
<b>“in response to finding said discrete module in said storage area, providing module information relating to said discrete module to said first program.”</b>	<p>The “said discrete module” is identified in Claim 1, Preamble.</p> <p>The “said storage area” is identified in Claim 2.</p> <p>The “module information” is identified in Claim 1, Element (b).</p> <p>The “said first program” is identified in Claim 1, Preamble.</p> <p>The “providing module information relating to said discrete module to said first program” occurs when one or more portions of the Dalvik VM transmit, or aid or manage the transmission of, “module information” (identified above) concerning the “discrete module” (identified above) to a location accessible by the “first program” (identified above).</p> <p>The “providing” occurs “in response to finding said discrete module in said storage area” – <i>i.e.</i>, after, or as a result of, the “searching for said discrete module in a storage area” (identified in Claim 2 above) arriving at, or locating, the “discrete module” (identified above).</p> <p>Function <code>dvmResolveClass</code> in file <code>platform_dalvik-master\vm\oo\Resolve.cpp</code> is called by the first program to find the discrete module in storage area (see Claim 2 analysis). When the discrete module is found, function <code>dvmResolveClass</code> returns module information to the first program via the <code>ClassObject</code> data structure <code>resClass</code>:</p> <pre>ClassObject * <b>dvmResolveClass</b>(const ClassObject* referrer, u4 classIdx, <b>bool</b> fromUnverifiedConstant) { DvmDex* pDvmDex = referrer-&gt;pDvmDex; ClassObject* resClass;  . .</pre>

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

	<pre>         return resClass;     } </pre> <p>The ClassObject data structure by which module information is transmitted back to the first program is defined in file</p> <p>platform_dalvik-master\vm\oo\Object.h :</p> <pre> /*  * Class objects have many additional fields. This is used for  both  * classes and interfaces, including synthesized classes  (arrays and  * primitive types).  *  * Class objects are unusual in that they have some fields  allocated with  * the system malloc (or LinearAlloc), rather than on the GC  heap. This is  * handy during initialization, but does require special  handling when  * discarding java.lang.Class objects.  *  * The separation of methods (direct vs. virtual) and fields  (class vs.  * instance) used in Dalvik works out pretty well. The only  time it's  * annoying is when enumerating or searching for things with  reflection.  */ <b>struct</b> ClassObject : Object {     /* leave space for instance data; we could access fields  directly if we     freeze the definition of java/lang/Class */     u4          instanceData[CLASS_FIELD_SLOTS];      /* UTF-8 descriptor for the class; from constant pool, or on  heap     if generated ("[C") */     <b>const char*</b> descriptor;     <b>char*</b>        descriptorAlloc;      /* access flags; low 16 bits are defined by VM spec */     u4          accessFlags;      /* VM-unique class serial number, nonzero, set very early  */     u4          serialNumber; </pre>
--	--

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

	<pre> /* DexFile from which we came; needed to resolve constant pool entries */ /* (will be NULL for VM-generated, e.g. arrays and primitive classes) */ DvmDex*      pDvmDex;  /* state of class initialization */ ClassStatus  status;  /* if class verify fails, we must return same error on subsequent tries */ ClassObject* verifyErrorClass;  /* threadId, used to check for recursive &lt;clinit&gt; invocation */ u4          initThreadId;  /*  * Total object size; used when allocating storage on gc heap. (For  * interfaces and abstract classes this will be zero.)  */ size_t      objectSize;  /* arrays only: class object for base element, for instanceof/checkcast (for String[][][], this will be String) */ ClassObject* elementClass;  /* arrays only: number of dimensions, e.g. int[][] is 2 */ int         arrayDim;  /* primitive type index, or PRIM_NOT (-1); set for generated prim classes */ PrimitiveType primitiveType;  /* superclass, or NULL if this is java.lang.Object */ ClassObject* super;  /* defining class loader, or NULL for the "bootstrap" system loader */ Object*     classLoader;  /* initiating class loader list */ /* NOTE: for classes with low serialNumber, these are unused, and the values are kept in a table in gDvm. */ InitiatingLoaderList initiatingLoaderList;  /* array of interfaces this class implements directly */ int         interfaceCount; </pre>
--	--

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

	<pre> ClassObject** interfaces;  /* static, private, and &lt;init&gt; methods */ <b>int</b>         directMethodCount; Method*      directMethods;  /* virtual methods defined in this class; invoked through vtable */ <b>int</b>         virtualMethodCount; Method*      virtualMethods;  /*  * Virtual method table (vtable), for use by "invoke-virtual". The  * vtable from the superclass is copied in, and virtual methods from  * our class either replace those from the super or are appended.  */ <b>int</b>         vtableCount; Method**      vtable;  /*  * Interface table (iftable), one entry per interface supported by  * this class. That means one entry for each interface we support  * directly, indirectly via superclass, or indirectly via  * superinterface. This will be null if neither we nor our superclass  * implement any interfaces.  *  * Why we need this: given "class Foo implements Face", declare  * "Face faceObj = new Foo()". Invoke faceObj.blah(), where "blah" is  * part of the Face interface. We can't easily use a single vtable.  *  * For every interface a concrete class implements, we create a list of  * virtualMethod indices for the methods in the interface.  */ <b>int</b>         iftableCount; InterfaceEntry* iftable;  /*  * The interface vtable indices for iftable get stored here. By placing  * them all in a single pool for each class that implements interfaces, </pre>
--	---

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

	<pre> * we decrease the number of allocations. */ <b>int</b>      ifviPoolCount; <b>int*</b>     ifviPool;  /* instance fields * * These describe the layout of the contents of a DataObject-compatible * Object. Note that only the fields directly defined by this class * are listed in ifields; fields defined by a superclass are listed * in the superclass's ClassObject.ifields. * * All instance fields that refer to objects are guaranteed to be * at the beginning of the field list. ifieldRefCount specifies * the number of reference fields. */ <b>int</b>      ifieldCount; <b>int</b>      ifieldRefCount; // number of fields that are object refs InstField* ifields;  /* bitmap of offsets of ifields */ u4 refOffsets;  /* source file name, if known */ <b>const char*</b> sourceFile;  /* static fields */ <b>int</b>      sfieldCount; StaticField sfields[0]; /* MUST be last item */ }; </pre> <p>In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.</p>
--	--

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

<b>Claim 6</b>	
<b>Claim Language</b>	<b>Analysis</b>
<b>(Preamble) “The method of claim 2 further comprising the step of,”</b>	<i>See</i> Claim 2 analysis.
<b>“in response to not finding said discrete module in said storage area, providing an error message in said response.”</b>	<p>The “said discrete module” is identified in Claim 1, Preamble.</p> <p>The “said storage area” is identified in Claim 2.</p> <p>The “error message” is a message indicating that the discrete module has not been found.</p> <p>The “providing an error message” occurs when one or more portions of the Dalvik VM cause, aid, or manage a transmission that includes an “error message” (identified above). <i>See, e.g.</i>, function <code>dvmResolveClass</code> in file <code>platform_dalvik-master\vm\oo\Resolve.cpp</code>:</p> <pre>     if (resClass != NULL) {          .         .          /*          * Add what we found to the list so we can skip the          class search          * next time through.          */          .         .      } else {         /* not found, exception should be raised */         LOGVV("Class not found: %s",             dexStringByTypeIdx(pDvmDex-&gt;pDexFile,             classIdx));         assert(dvmCheckException(dvmThreadSelf()));     } </pre> <p>The “providing” occurs “in response to not finding said discrete module in said storage area” – <i>i.e.</i>, after, or as a result of, the “searching for said discrete module in a storage area” (identified in Claim 2 above) not arriving at, or not locating, the “discrete module” (identified above).</p>

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

	<p>In the “else” clause of above code, the discrete module has not been found in the storage area, <i>i.e.</i>, resClass = NULL, so an exception (error) is raised. A string error message is written to a log (LOGVV “Class not found”) and dvmCheckException is asserted, which when handled by Dalvik VM in conjunction with Android OS may result in an appropriate action in response to the error message.</p> <p>The claim language “in said response” refers back to the claim language “in response to finding”. The error message is “in said response” – <i>i.e.</i>, it is in the transmission that occurs after, or as a result of, the “searching for said discrete module in a storage area” (identified in Claim 2 above) not arriving at, or locating, the “discrete module” (identified above).</p> <p>In the alternative, the Accused Systems meet this element under the doctrine of equivalents, because the accused components identified above perform substantially the same function in substantially the same way to obtain substantially the same result.</p>
--	--

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY****ENDNOTES**


---

<sup>i</sup> platform\_dalvik-master\libdex\DexFile.h

```

/*
 * Access .dex (Dalvik Executable Format) files. The code here assumes that
 * the DEX file has been rewritten (byte-swapped, word-aligned) and that
 * the contents can be directly accessed as a collection of C arrays. Please
 * see docs/dalvik/dex-format.html for a detailed description.
 *
 * The structure and field names were chosen to match those in the DEX spec.
 *
 * It's generally assumed that the DEX file will be stored in shared memory,
 * obviating the need to copy code and constant pool entries into newly
 * allocated storage. Maintaining local pointers to items in the shared area
 * is valid and encouraged.
 *
 * All memory-mapped structures are 32-bit aligned unless otherwise noted.
 */

/*
 * Direct-mapped "header_item" struct.
 */
structDexHeader {
    u1magic[8];          /* includes version number */
    u4checksum;          /* Adler32 checksum */
    u1signature[kSHA1DigestLen]; /* SHA-1 hash */
    u4fileSize;          /* length of entire file */
    u4headerSize;        /* offset to start of next section */
    u4endianTag;
    u4linkSize;
    u4linkOff;
    u4mapOff;
    u4stringIdsSize;
    u4stringIdsOff;
    u4typeIdsSize;
    u4typeIdsOff;
    u4protoIdsSize;
    u4protoIdsOff;
    u4fieldIdsSize;
    u4fieldIdsOff;
    u4methodIdsSize;
    u4methodIdsOff;
    u4classDefsSize;
    u4classDefsOff;
    u4dataSize;
    u4dataOff;
};

/*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * Direct-mapped "map_item".
    */
    struct DexMapItem {
        u2type;          /* type code (see kDexType* above) */
        u2unused;
        u4size;          /* count of items of the indicated type */
        u4offset;        /* file offset to the start of data */
    };

    /*
    * Direct-mapped "map_list".
    */
    struct DexMapList {
        u4size;          /* #of entries in list */
        DexMapItemlist[1]; /* entries */
    };

    /*
    * Direct-mapped "string_id_item".
    */
    struct DexStringId {
        u4stringDataOff; /* file offset to string_data_item */
    };

    /*
    * Direct-mapped "type_id_item".
    */
    struct DexTypeId {
        u4descriptorIdx; /* index into stringIds list for type descriptor */
    };

    /*
    * Direct-mapped "field_id_item".
    */
    struct DexFieldId {
        u2classIdx;      /* index into typelds list for defining class */
        u2typeldx;        /* index into typelds for field type */
        u4nameldx;        /* index into stringIds for field name */
    };

    /*
    * Direct-mapped "method_id_item".
    */
    struct DexMethodId {
        u2classIdx;      /* index into typelds list for defining class */
        u2protoldx;       /* index into protolds for method prototype */
        u4nameldx;        /* index into stringIds for method name */
    };

    /*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * Direct-mapped "proto_id_item".
    */
    struct DexProtoId {
        u4shortyIdx;      /* index into stringIds for shorty descriptor */
        u4returnTypeIdx;  /* index into typeIds list for return type */
        u4parametersOff;  /* file offset to type_list for parameter types */
    };

    /*
    * Direct-mapped "class_def_item".
    */
    struct DexClassDef {
        u4classIdx;      /* index into typeIds for this class */
        u4accessFlags;
        u4superclassIdx; /* index into typeIds for superclass */
        u4interfacesOff; /* file offset to DexTypeList */
        u4sourceFileIdx; /* index into stringIds for source file name */
        u4annotationsOff; /* file offset to annotations_directory_item */
        u4classDataOff;  /* file offset to class_data_item */
        u4staticValuesOff; /* file offset to DexEncodedArray */
    };

    /*
    * Direct-mapped "type_item".
    */
    struct DexTypeItem {
        u2typeIdx;      /* index into typeIds */
    };

    /*
    * Direct-mapped "type_list".
    */
    struct DexTypeList {
        u4size;          /* #of entries in list */
        DexTypeItemlist[1]; /* entries */
    };

    /*
    * Direct-mapped "code_item".
    *
    * The "catches" table is used when throwing an exception,
    * "debugInfo" is used when displaying an exception stack trace or
    * debugging. An offset of zero indicates that there are no entries.
    */
    struct DexCode {
        u2registersSize;
        u2insSize;
        u2outsSize;
        u2triesSize;
        u4debugInfoOff; /* file offset to debug info stream */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

u4insnsSize;      /* size of the insns array, in u2 units */
u2insns[1];
/* followed by optional u2 padding */
/* followed by try_item[triesSize] */
/* followed by uleb128 handlersSize */
/* followed by catch_handler_item[handlersSize] */
};

/*
 * Direct-mapped "try_item".
 */
structDexTry {
u4startAddr;      /* start address, in 16-bit code units */
u2insnCount;      /* instruction count, in 16-bit code units */
u2handlerOff;     /* offset in encoded handler data to handlers */
};

/*
 * Link table. Currently undefined.
 */
structDexLink {
u1bleargh;
};

/*
 * Direct-mapped "annotations_directory_item".
 */
structDexAnnotationsDirectoryItem {
u4classAnnotationsOff; /* offset to DexAnnotationSetItem */
u4fieldsSize;          /* count of DexFieldAnnotationsItem */
u4methodsSize;         /* count of DexMethodAnnotationsItem */
u4parametersSize;      /* count of DexParameterAnnotationsItem */
/* followed by DexFieldAnnotationsItem[fieldsSize] */
/* followed by DexMethodAnnotationsItem[methodsSize] */
/* followed by DexParameterAnnotationsItem[parametersSize] */
};

/*
 * Direct-mapped "field_annotations_item".
 */
structDexFieldAnnotationsItem {
u4fieldIdx;
u4annotationsOff;    /* offset to DexAnnotationSetItem */
};

/*
 * Direct-mapped "method_annotations_item".
 */
structDexMethodAnnotationsItem {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

u4methodIdx;
u4annotationsOff;          /* offset to DexAnnotationSetItem */
};

/*
 * Direct-mapped "parameter_annotations_item".
 */
structDexParameterAnnotationsItem {
u4methodIdx;
u4annotationsOff;          /* offset to DexAnnotationSetRefList */
};

/*
 * Direct-mapped "annotation_set_ref_item".
 */
structDexAnnotationSetRefItem {
u4annotationsOff;          /* offset to DexAnnotationSetItem */
};

/*
 * Direct-mapped "annotation_set_ref_list".
 */
structDexAnnotationSetRefList {
u4size;
DexAnnotationSetRefItemlist[1];
};

/*
 * Direct-mapped "annotation_set_item".
 */
structDexAnnotationSetItem {
u4size;
u4entries[1];              /* offset to DexAnnotationItem */
};

/*
 * Direct-mapped "annotation_item".
 *
 * NOTE: this structure is byte-aligned.
 */
structDexAnnotationItem {
u1visibility;
u1annotation[1];           /* data in encoded_annotation format */
};

/*
 * Direct-mapped "encoded_array".
 *
 * NOTE: this structure is byte-aligned.
 */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

structDexEncodedArray {
    u1array[1];          /* data in encoded_array format */
};

```

```

ii \android-platform_frameworks_base-57ea96a\cmds\app_process\app_main.cpp
    intmain(intargc, constchar* constargv[])
    {

```

```

        AppRuntime runtime;

```

```

        if (zygote) {
            runtime.start("com.android.internal.os.ZygoteInit",
                startSystemServer ? "start-system-server" : "");
            elseif (className) {
                // Remainder of args get passed to startup class main()
                runtime.mClassName = className;
                runtime.mArgC = argc - i;
                runtime.mArgV = argv + i;
                runtime.start("com.android.internal.os.RuntimeInit",
                    application ? "application" : "tool");
            } else {
                fprintf(stderr, "Error: no class name or --zygote supplied.\n");
                app_usage();
                LOG_ALWAYS_FATAL("app_process: no class name or --zygote supplied.");
                return 10;
            }

```

```

iii \android-platform_frameworks_base-
57ea96a\core\java\com\android\internal\os\ZygoteInit.java (1/8/2012)
/**

```

```

 * Startup class for the zygote process.
 *
 * Pre-initializes some classes, and then waits for commands on a UNIX domain
 * socket. Based on these commands, forks off child processes that inherit
 * the initial state of the VM.
 *
 * Please see {@link ZygoteConnection.Arguments} for documentation on the
 * client protocol.
 *
 * @hide
 */

```

```

publicclassZygoteInit {

```

```

    publicstaticvoidmain(String argv[]) {
    try {

```

```

        registerZygoteSocket();

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        preload();

// Do an initial gc to clean up after startup
gc();

// If requested, start system server directly from Zygote
if (argv.length != 2) {
    throw new RuntimeException(argv[0] + USAGE_STRING);
}

if (argv[1].equals("start-system-server")) {
    startSystemServer();
} elseif (!argv[1].equals("")) {
    throw new RuntimeException(argv[0] + USAGE_STRING);
}

Log.i(TAG, "Accepting command socket connections");
}

static void preload() {
    preloadClasses();
    preloadResources();
}

    \android-platform_frameworks_base-
57ea96a\core\java\com\android\internal\os\ZygoteInit.java (1/8/2012)
/**
 * Performs Zygote process initialization. Loads and initializes
 * commonly used classes.
 *
 * Most classes only cause a few hundred bytes to be allocated, but
 * a few will allocate a dozen Kbytes (in one case, 500+K).
 */
private static void preloadClasses() {
    final VMRuntime runtime = VMRuntime.getRuntime();

    InputStream is = ZygoteInit.class.getClassLoader().getResourceAsStream(
        PRELOADED_CLASSES);
    if (is == null) {
        Log.e(TAG, "Couldn't find " + PRELOADED_CLASSES + ".");
    } else {
        Log.i(TAG, "Preloading classes...");
        long startTime = SystemClock.uptimeMillis();

        try {
            BufferedReader br
                = new BufferedReader(new InputStreamReader(is), 256);

            int count = 0;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        String line;
while ((line = br.readLine()) != null) {
// Skip comments and blank lines.
        line = line.trim();
if (line.startsWith("#") || line.equals("")) {
continue;
        }

try {
if (false) {
Log.v(TAG, "Preloading " + line + "...");
        }
Class.forName(line);

        \android-platform_frameworks_base-
        57ea96a\core\java\com\android\internal\os\ZygoteInit.java (1/8/2012)
/**
 * The name of a resource file that contains classes to preload.
 */
private static final String PRELOADED_CLASSES = "preloaded-classes";

        \android-platform_frameworks_base-57ea96a\preloaded-classes
(edited-2300 classes listed)(1/8/2012)
# Classes which are preloaded by com.android.internal.os.ZygoteInit.
# Automatically generated by frameworks/base/tools/preload/WritePreloadedClassFile.java.
# MIN_LOAD_TIME_MICROS=1250
# MIN_PROCESSES=10
java.lang.Boolean
java.lang.BootClassLoader
java.lang.Byte
java.lang.CaseMapper
java.lang.CharSequence
java.lang.Character
java.lang.Character$UnicodeBlock
java.lang.Class
java.lang.System
java.lang.Thread
java.lang.Thread$State
java.lang.Thread$UncaughtExceptionHandler
java.lang.ThreadGroup
java.lang.ThreadLocal
java.lang.ThreadLocal$Values
java.lang.Throwable
java.lang.TypeNotPresentException
java.lang.UnsafeByteSequence
java.lang.UnsatisfiedLinkError
java.lang.UnsupportedOperationException
java.lang.VMClassLoader
java.lang.VMThread

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

iv Source code not pasted to maintain some brevity (see appendix for source code): :  
frameworks/base/core/java/com/android/internal/os/ZygoteConnection.java

v Source code not pasted to maintain some brevity (see appendix for source code): :  
libcore/dalvik/src/main/java/dalvik/system/Zygote.java

vi \android-platform\_frameworks\_base-57ea96a\cmds\app\_process\app\_main.cpp  
**int**main(intargc, **const**char\* constargv[])  
{

AppRuntime runtime;

```

if (zygote) {
runtime.start("com.android.internal.os.ZygoteInit",
startSystemServer ?"start-system-server" : "");
} elseif (className) {
// Remainder of args get passed to startup class main()
runtime.mClassName = className;
runtime.mArgC = argc - i;
runtime.mArgV = argv + i;
runtime.start("com.android.internal.os.RuntimeInit",
application ?"application" : "tool");
} else {
fprintf(stderr, "Error: no class name or --zygote supplied.\n");
app_usage();
LOG_ALWAYS_FATAL("app_process: no class name or --zygote supplied.");
return 10;
}

```

vii \android-platform\_frameworks\_base-57ea96a\core\jni\AndroidRuntime.cpp (1/8/2012)

```

/*
 * Start the Android runtime. This involves starting the virtual machine
 * and calling the "static void main(String[] args)" method in the class

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* named by "className".
*
* Passes the main function two arguments, the class name and the specified
* options string.
*/
void AndroidRuntime::start(constchar* className, constchar* options)
{
/* start the virtual machine */
JNIEnv* env;
if (startVm(&mJavaVM, &env) != 0) {
return;
}
onVmCreated(env);

/*
* Register android functions.
*/
if (startReg(env) < 0) {
    LOGE("Unable to register all android natives\n");
return;
}
}

/*
* Start the Dalvik Virtual Machine.
*
* Various arguments, most determined by system properties, are passed in.
* The "mOptions" vector is updated.
*
* Returns 0 on success.
*/
int AndroidRuntime::startVm(JavaVM** pJavaVM, JNIEnv** pEnv)
{
int result = -1;
JavaVMInitArgs initArgs;
JavaVMOption opt;
char propBuf[PROPERTY_VALUE_MAX];
char stackTraceFileBuf[PROPERTY_VALUE_MAX];
char dexoptFlagsBuf[PROPERTY_VALUE_MAX];
char enableAssertBuf[sizeof(" -ea:") - 1 + PROPERTY_VALUE_MAX];
char jniOptsBuf[sizeof(" -Xjniopts:") - 1 + PROPERTY_VALUE_MAX];
char heapstartSizeOptsBuf[sizeof(" -Xms") - 1 + PROPERTY_VALUE_MAX];
char heapSizeOptsBuf[sizeof(" -Xmx") - 1 + PROPERTY_VALUE_MAX];
char heapGrowthLimitOptsBuf[sizeof(" -XX:HeapGrowthLimit=") - 1 + PROPERTY_VALUE_MAX];
char extraOptsBuf[PROPERTY_VALUE_MAX];
char* stackTraceFile = NULL;
bool checkJni = false;
bool checkDexSum = false;
bool logStdio = false;
enum {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

kEMDefault,
kEMIntPortable,
kEMIntFast,
#ifdef(WITH_JIT)
kEMJitCompiler,
#endif
    } executionMode = kEMDefault;

property_get("dalvik.vm.execution-mode", propBuf, "");
if (strcmp(propBuf, "int:portable") == 0) {
    executionMode = kEMIntPortable;
} elseif (strcmp(propBuf, "int:fast") == 0) {
    executionMode = kEMIntFast;
#ifdef(WITH_JIT)
} elseif (strcmp(propBuf, "int:jit") == 0) {
    executionMode = kEMJitCompiler;
#endif
}

/*
 * Initialize the VM.
 *
 * The JavaVM* is essentially per-process, and the JNIEnv* is per-thread.
 * If this call succeeds, the VM is ready, and we can start issuing
 * JNI calls.
 */
if (JNI_CreateJavaVM(pJavaVM, pEnv, &initArgs) < 0) {
    LOGE("JNI_CreateJavaVM failed\n");
    goto bail;
}

    result = 0;

bail:
    free(stackTraceFile);
return result;
}

viii      \platform_dalvik-master\vm\Jni.cpp

/*
 * Create a new VM instance.
 *
 * The current thread becomes the main VM thread. We return immediately,

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * which effectively means the caller is executing in a native method.
    */
jint JNI_CreateJavaVM(JavaVM** p_vm, JNIEnv** p_env, void* vm_args) {
    constJavaVMInitArgs* args = (JavaVMInitArgs*) vm_args;

    /*
     * Set up structures for JNIEnv and VM.
     */
    JavaVMExt* pVM = (JavaVMExt*) calloc(1, sizeof(JavaVMExt));
    pVM->funcTable = &gInvokeInterface;
    pVM->envList = NULL;
    dvmInitMutex(&pVM->envListLock);

    UniquePtr<constchar*>argv(newconstchar*[args->nOptions]);
    memset(argv.get(), 0, sizeof(char*) * (args->nOptions));

    /*
     * Create a JNIEnv for the main thread. We need to have something set up
     * here because some of the class initialization we do when starting
     * up the VM will call into native code.
     */
    JNIEnvExt* pEnv = (JNIEnvExt*) dvmCreateJNIEnv(NULL);

    /* Initialize VM. */
    gDvm.initializing = true;
    std::string status =
        dvmStartup(argc, argv.get(), args->ignoreUnrecognized, (JNIEnv*)pEnv);
    gDvm.initializing = false;

    if (!status.empty()) {
        free(pEnv);
        free(pVM);
        ALOGW("CreateJavaVM failed: %s", status.c_str());
        return JNI_ERR;
    }

    /*
     * Success! Return stuff to caller.
     */
    dvmChangeStatus(NULL, THREAD_NATIVE);
    *p_env = (JNIEnv*) pEnv;
    *p_vm = (JavaVM*) pVM;
    ALOGV("CreateJavaVM succeeded");
    return JNI_OK;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

ix      platform_dalvik-master\vm\Init.cpp

/*
 * VM initialization. Pass in any options provided on the command line.
 * Do not pass in the class name or the options for the class.
 *
 * Returns 0 on success.
 */
std::string dvmStartup(int argc, constchar* constargv[],
boolignoreUnrecognized, JNIEnv* pEnv)
{
    ScopedShutdownscopedShutdown;

    assert(gDvm.initializing);

    ALOGV("VM initargs (%d):", argc);
    for (inti = 0; i<argc; i++) {
        ALOGV(" %d: '%s'", i, argv[i]);
    }
    setCommandLineDefaults();

/*
 * Process the option flags (if any).
 */
int cc = processOptions(argc, argv, ignoreUnrecognized);
if (cc != 0) {
if (cc < 0) {
    dvmFprintf(stderr, "\n");
    usage("dalvikvm");
}
return"syntax error";
}

/* mterp setup */
    ALOGV("Using executionMode %d", gDvm.executionMode);
    dvmCheckAsmConstants();

/*
 * Initialize components.
 */
    dvmQuasiAtomicsStartup();
    if (!dvmAllocTrackerStartup()) {
return"dvmAllocTrackerStartup failed";
    }
    if (!dvmGcStartup()) {
return"dvmGcStartup failed";
    }
    if (!dvmThreadStartup()) {
return"dvmThreadStartup failed";
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

if (!dvmInlineNativeStartup()) {
return"dvmInlineNativeStartup";
}
if (!dvmRegisterMapStartup()) {
return"dvmRegisterMapStartup failed";
}
if (!dvmInstanceofStartup()) {
return"dvmInstanceofStartup failed";
}
if (!dvmClassStartup()) {
return "dvmClassStartup failed";
}

/*
 * At this point, the system is guaranteed to be sufficiently
 * initialized that we can look up classes and class members. This
 * call populates the gDvm instance with all the class and member
 * references that the VM wants to use directly.
 */
if (!dvmFindRequiredClassesAndMembers()) {
return"dvmFindRequiredClassesAndMembers failed";
}

if (!dvmStringInternStartup()) {
return"dvmStringInternStartup failed";
}
if (!dvmNativeStartup()) {
return"dvmNativeStartup failed";
}
if (!dvmInternalNativeStartup()) {
return"dvmInternalNativeStartup failed";
}
if (!dvmJniStartup()) {
return"dvmJniStartup failed";
}
if (!dvmProfilingStartup()) {
return"dvmProfilingStartup failed";
}

/*
 * Create a table of methods for which we will substitute an "inline"
 * version for performance.
 */
if (!dvmCreateInlineSubsTable()) {
return"dvmCreateInlineSubsTable failed";
}

/*
 * Miscellaneous class library validation.
 */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

if (!dvmValidateBoxClasses()) {
return"dvmValidateBoxClasses failed";
}

/*
 * Do the last bits of Thread struct initialization we need to allow
 * JNI calls to work.
 */
if (!dvmPrepMainForJni(pEnv)) {
return"dvmPrepMainForJni failed";
}

/*
 * Explicitly initialize java.lang.Class. This doesn't happen
 * automatically because it's allocated specially (it's an instance
 * of itself). Must happen before registration of system natives,
 * which make some calls that throw assertions if the classes they
 * operate on aren't initialized.
 */
if (!dvmInitClass(gDvm.classJavaLangClass)) {
return"couldn't initialized java.lang.Class";
}

/*
 * Register the system native methods, which are registered through JNI.
 */
if (!registerSystemNatives(pEnv)) {
return"couldn't register system natives";
}

/*
 * Do some "late" initialization for the memory allocator. This may
 * allocate storage and initialize classes.
 */
if (!dvmCreateStockExceptions()) {
return"dvmCreateStockExceptions failed";
}

/*
 * At this point, the VM is in a pretty good state. Finish prep on
 * the main thread (specifically, create a java.lang.Thread object to go
 * along with our Thread struct). Note we will probably be executing
 * some interpreted class initializer code in here.
 */
if (!dvmPrepMainThread()) {
return"dvmPrepMainThread failed";
}

/*
 * Make sure we haven't accumulated any tracked references. The main

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * thread should be starting with a clean slate.
    */
if (dvmReferenceTableEntries(&dvmThreadSelf()->internalLocalRefTable) != 0)
{
    ALOGW("Warning: tracked references remain post-initialization");
    dvmDumpReferenceTable(&dvmThreadSelf()->internalLocalRefTable, "MAIN");
}

/* general debugging setup */
if (!dvmDebuggerStartup()) {
    return "dvmDebuggerStartup failed";
}

if (!dvmGcStartupClasses()) {
    return "dvmGcStartupClasses failed";
}

/*
    * Init for either zygote mode or non-zygote mode. The key difference
    * is that we don't start any additional threads in Zygote mode.
    */
if (gDvm.zygote) {
    if (!initZygote()) {
        return "initZygote failed";
    }
} else {
    if (!dvmInitAfterZygote()) {
        return "dvmInitAfterZygote failed";
    }
}

#ifdef NDEBUG
if (!dvmTestHash())
    ALOGE("dvmTestHash FAILED");
if (false/*noisy!*/&& !dvmTestIndirectRefTable())
    ALOGE("dvmTestIndirectRefTable FAILED");
#endif

if (dvmCheckException(dvmThreadSelf())) {
    dvmLogExceptionStackTrace();
    return "Exception pending at end of VM initialization";
}

scopedShutdown.disarm();
return "";
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

x        \platform\_dalvik-master\vm\oo\Class.h (8/30/2011)

```

/*
 * The classpath and bootclasspath differ in that only the latter is
 * consulted when looking for classes needed by the VM. When searching
 * for an arbitrary class definition, we start with the bootclasspath,
 * look for optional packages (a/k/a standard extensions), and then try
 * theclasspath.
 *
 * In Dalvik, a class can be found in one of two ways:
 * - in a .dex file
 * - in a .dex file named specifically "classes.dex", which is held
 *   inside a jar file
 *
 * These two may be freely intermixed in a classpath specification.
 * Ordering is significant.
 */

```

xi        \platform\_dalvik-master\vm\oo\Class.cpp (10/30/2013)

```

/*
 * Initialize the bootstrap class loader.
 *
 * Call this after the bootclasspath string has been finalized.
 */
booldvmClassStartup()
{
/* make this a requirement -- don't currently support dirs in path */
if (strcmp(gDvm.bootClassPathStr, ".") == 0) {
    ALOGE("ERROR: must specify non-'.' bootclasspath");
return false;
}

gDvm.loadedClasses =
dvmHashTableCreate(256, (HashFreeFunc) dvmFreeClassInnards);

gDvm.pBootLoaderAlloc = dvmLinearAllocCreate(NULL);
if (gDvm.pBootLoaderAlloc == NULL)
return false;

if (false) {
    linearAllocTests();
    exit(0);
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/*
 * Class serial number. We start with a high value to make it distinct
 * in binary dumps (e.g. hprof).
 */
gDvm.classSerialNumber = INITIAL_CLASS_SERIAL_NUMBER;

/*
 * Set up the table we'll use for tracking initiating loaders for
 * early classes.
 * If it's NULL, we just fall back to the InitiatingLoaderList in the
 * ClassObject, so it's not fatal to fail this allocation.
 */
gDvm.initiatingLoaderList = (InitiatingLoaderList*)
calloc(ZYGOTE_CLASS_CUTOFF, sizeof(InitiatingLoaderList));

/*
 * Create the initial classes. These are the first objects constructed
 * within the nascent VM.
 */
if (!createInitialClasses()) {
returnfalse;
}

/*
 * Process the bootstrap class path. This means opening the specified
 * DEX or Jar files and possibly running them through the optimizer.
 */
assert(gDvm.bootClassPath == NULL);
processClassPath(gDvm.bootClassPathStr, true);

if (gDvm.bootClassPath == NULL)
returnfalse;

returntrue;
}

```

xii      \platform\_dalvik-master\vm\oo\Class.cpp (10/30/2013)

```

/*
 * Convert a colon-separated list of directories, Zip files, and DEX files
 * into an array of ClassPathEntrystructs.
 *
 * During normal startup we fail if there are no entries, because we won't
 * get very far without the basic language support classes, but if we're
 * optimizing a DEX file we allow it.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

*
* If entries are added or removed from the bootstrap class path, the
* dependencies in the DEX files will break, and everything except the
* very first entry will need to be regenerated.
*/
staticClassPathEntry* processClassPath(constchar* pathStr, boolisBootstrap)
{
ClassPathEntry* cpe = NULL;
char* mangle;
char* cp;
constchar* end;
intidx, count;

    assert(pathStr != NULL);

    mangle = strdup(pathStr);

/*
    * Allocate storage. We over-alloc by one so we can set an "end" marker.
    */
cpe = (ClassPathEntry*) calloc(count+1, sizeof(ClassPathEntry));

/*
    * Set the global pointer so the DEX file dependency stuff can find it.
    */
gDvm.bootClassPath = cpe;

/*
    * Go through a second time, pulling stuff out.
    */
cp = mangle;
idx = 0;
while (cp < end) {
if (*cp == '\\0') {
/* leading, trailing, or doubled ':'; ignore it */
} else {
if (isBootstrap&&
dvmPathToAbsolutePortion(cp) == NULL) {
    ALOGE("Non-absolute bootclasspath entry '%s'", cp);
    free(cpe);
cpe = NULL;
goto bail;
}

ClassPathEntrytmp;
tmp.kind = kCpeUnknown;
tmp.fileName = strdup(cp);
tmp.ptr = NULL;

/*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        * Drop an end marker here so DEX loader can walk unfinished
        * list.
        */
cpe[idx].kind = kCpeLastEntry;
cpe[idx].fileName = NULL;
cpe[idx].ptr = NULL;

if (!prepareCpe(&tmp, isBootstrap)) {
/* drop from list and continue on */
    free(tmp.fileName);
} else {
/* copy over, pointers and all */
cpe[idx] = tmp;
idx++;
}
}

cp += strlen(cp) + 1;
}
}

```

xiii      \platform\_dalvik-master\vm\oo\Class.cpp (10/30/2013)

```

/*
 * Prepare a ClassPathEntrystruct, which at this point only has a valid
 * filename. We need to figure out what kind of file it is, and for
 * everything other than directories we need to open it up and see
 * what's inside.
 */
static bool prepareCpe(ClassPathEntry* cpe, bool isBootstrap)
{
struct statsb;

if (stat(cpe->fileName, &sb) < 0) {
    ALOGD("Unable to stat classpath element '%s'", cpe->fileName);
return false;
}
if (S_ISDIR(sb.st_mode)) {
    ALOGE("Directory classpath elements are not supported: %s", cpe->fileName);
return false;
}

char suffix[10];
getFileNameSuffix(cpe->fileName, suffix, sizeof(suffix));

if ((strcmp(suffix, "jar") == 0) || (strcmp(suffix, "zip") == 0) ||

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        (strcmp(suffix, "apk") == 0)) {
JarFile* pJarFile = NULL;
if (dvmJarFileOpen(cpe->fileName, NULL, &pJarFile, isBootstrap) == 0) {
cpe->kind = kCpeJar;
cpe->ptr = pJarFile;
return true;
}
} elseif (strcmp(suffix, "dex") == 0) {
RawDexFile* pRawDexFile = NULL;
if (dvmRawDexFileOpen(cpe->fileName, NULL, &pRawDexFile, isBootstrap) == 0) {
cpe->kind = kCpeDex;
cpe->ptr = pRawDexFile;
return true;
}
} else {
ALOG("Unknown type suffix '%s'", suffix);
}

ALOGD("Unable to process classpath element '%s'", cpe->fileName);
return false;
}

```

xiv      \platform\_dalvik-master\vm\JarFile.cpp

```

/*
 * Access the contents of a Jar file.
 *
 * This isn't actually concerned with any of the Jar-like elements; it
 * just wants a zip archive with "classes.dex" inside. In Android the
 * most common example is ".apk".
 */

/*
 * Open a Jar file. It's okay if it's just a Zip archive without all of
 * the Jar trimmings, but we do insist on finding "classes.dex" inside
 * or an appropriately-named ".odex" file alongside.
 *
 * If "isBootstrap" is not set, the optimizer/verifier regards this DEX as
 * being part of a different class loader.
 */
int dvmJarFileOpen(constchar* fileName, constchar* odexOutputName,
JarFile** ppJarFile, bool isBootstrap)
{
/*
 * TODO: This function has been duplicated and modified to become
 * dvmRawDexFileOpen() in RawDexFile.c. This should be refactored.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

*/

ZipArchive archive;
DvmDex* pDvmDex = NULL;
char* cachedName = NULL;
bool archiveOpen = false;
bool locked = false;
int fd = -1;
int result = -1;

/* Even if we're not going to look at the archive, we need to
 * open it so we can stuff it into ppJarFile.
 */
if (dexZipOpenArchive(fileName, &archive) != 0)
goto bail;
archiveOpen = true;

/* If we fork/exec into dexopt, don't let it inherit the archive's fd.
 */
dvmSetCloseOnExec(dexZipGetArchiveFd(&archive));

/* First, look for a ".odex" alongside the jar file. It will
 * have the same name/path except for the extension.
 */
fd = openAlternateSuffix(fileName, "odex", O_RDONLY, &cachedName);
if (fd >= 0) {
    ALOGV("Using alternate file (odex) for %s ...", fileName);
    if (!dvmCheckOptHeaderAndDependencies(fd, false, 0, 0, true, true)) {
        ALOGE("%s odex has stale dependencies", fileName);
        free(cachedName);
        cachedName = NULL;
        close(fd);
        fd = -1;
    }
    gototryArchive;
} else {
    ALOGV("%s odex has good dependencies", fileName);
//TODO: make sure that the .odex actually corresponds
//      to the classes.dex inside the archive (if present).
//      For typical use there will be no classes.dex.
}
} else {
ZipEntry entry;

tryArchive:
/*
 * Pre-created .odex absent or stale. Look inside the jar for a
 * "classes.dex".
 */
    entry = dexZipFindEntry(&archive, kDexInJarName);
    if (entry != NULL) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```
boolnewFile = false;
```

```
/*
```

```
    * We've found the one we want. See if there's an up-to-date copy
    * in the cache.
```

```
    *
```

```
    * On return, "fd" will be seeked just past the "opt" header.
```

```
    *
```

```
    * If a stale .odex file is present and classes.dex exists in
    * the archive, this will *not* return an fd pointing to the
    * .odex file; the fd will point into dalvik-cache like any
    * other jar.
```

```
    */
```

```
if (odexOutputName == NULL) {
```

```
    cachedName = dexOptGenerateCacheFileName(fileName,
    kDexInJarName);
```

```
    if (cachedName == NULL)
```

```
        goto bail;
```

```
    } else {
```

```
        cachedName = strdup(odexOutputName);
```

```
    }
```

```
    ALOGV("dvmJarFileOpen: Checking cache for %s (%s)",
    fileName, cachedName);
```

```
    fd = dvmOpenCachedDexFile(fileName, cachedName,
```

```
    dexGetZipEntryModTime(&archive, entry),
```

```
    dexGetZipEntryCrc32(&archive, entry),
```

```
    isBootstrap, &newFile, /*createIfMissing=*/true);
```

```
    if (fd < 0) {
```

```
        ALOGI("Unable to open or create cache for %s (%s)",
        fileName, cachedName);
```

```
        goto bail;
```

```
    }
```

```
    locked = true;
```

```
/*
```

```
    * If fd points to a new file (because there was no cached version,
    * or the cached version was stale), generate the optimized DEX.
```

```
    * The file descriptor returned is still locked, and is positioned
```

```
    * just past the optimization header.
```

```
    */
```

```
if (newFile) {
```

```
    u8startWhen, extractWhen, endWhen;
```

```
    bool result;
```

```
    off_tdexOffset;
```

```
    dexOffset = lseek(fd, 0, SEEK_CUR);
```

```
    result = (dexOffset > 0);
```

```
    if (result) {
```

```
        startWhen = dvmGetRelativeTimeUsec();
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        result = dexZipExtractEntryToFile(&archive, entry, fd) == 0;
extractWhen = dvmGetRelativeTimeUsec();
    }
    if (result) {
        result = dvmOptimizeDexFile(fd, dexOffset,
dexGetZipEntryUncompLen(&archive, entry),
fileName,
dexGetZipEntryModTime(&archive, entry),
dexGetZipEntryCrc32(&archive, entry),
isBootstrap);
    }

    if (!result) {
        ALOGE("Unable to extract+optimize DEX from '%s'",
fileName);
        goto bail;
    }

    endWhen = dvmGetRelativeTimeUsec();
        ALOGD("DEX prep '%s': unzip in %dms, rewrite %dms",
fileName,
            (int) (extractWhen - startWhen) / 1000,
            (int) (endWhen - extractWhen) / 1000);
    }
    } else {
        ALOGI("Zip is good, but no %s inside, and no valid .odex "
"file in the same directory", kDexInJarName);
        goto bail;
    }
}

/*
 * Map the cached version. This immediately rewinds the fd, so it
 * doesn't have to be seeked anywhere in particular.
 */
if (dvmDexFileOpenFromFd(fd, &pDvmDex) != 0) {
    ALOGI("Unable to map %s in %s", kDexInJarName, fileName);
    goto bail;
}

    ALOGV("Successfully opened '%s' in '%s'", kDexInJarName, fileName);

    *ppJarFile = (JarFile*) calloc(1, sizeof(JarFile));
    (*ppJarFile)->archive = archive;
    (*ppJarFile)->cacheFileName = cachedName;
    (*ppJarFile)->pDvmDex = pDvmDex;
    cachedName = NULL;    // don't free it below
    result = 0;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

xv      \platform_dalvik-master\vm\DvmDex.cpp

/*
 * Given an open optimized DEX file, map it into read-only shared memory and
 * parse the contents.
 *
 * Returns nonzero on error.
 */
int dvmDexFileOpenFromFd(int fd, DvmDex** ppDvmDex)
{
    DvmDex* pDvmDex;
    DexFile* pDexFile;
    MemMapping memMap;
    int parseFlags = kDexParseDefault;
    int result = -1;

    if (gDvm.verifyDexChecksum)
        parseFlags |= kDexParseVerifyChecksum;

    pDexFile = dexFileParse((u1*)memMap.addr, memMap.length, parseFlags);

    pDvmDex = allocateAuxStructures(pDexFile);

    /* tuck this into the DexFile so it gets released later */
    sysCopyMap(&pDvmDex->memMap, &memMap);
    pDvmDex->isMappedReadOnly = true;
    *ppDvmDex = pDvmDex;
    result = 0;
}

```

```

xvi      \platform_dalvik-master\libdex\DexFile.cpp

/*
 * Parse an optimized or unoptimized .dex file sitting in memory. This is
 * called after the byte-ordering and structure alignment has been fixed up.
 *
 * On success, return a newly-allocated DexFile.
 */
DexFile* dexFileParse(const u1* data, size_t length, int flags)
{
    DexFile* pDexFile = NULL;
    const DexHeader* pHeader;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

const u1* magic;
int result = -1;

if (length < sizeof(DexHeader)) {
    ALOGE("too short to be a valid .dex");
    goto bail;    /* bad file format */
}

    pDexFile = (DexFile*) malloc(sizeof(DexFile));
if (pDexFile == NULL)
    goto bail;    /* alloc failure */
    memset(pDexFile, 0, sizeof(DexFile));

/*
 * Peel off the optimized header.
 */
if (memcmp(data, DEX_OPT_MAGIC, 4) == 0) {
    magic = data;
if (memcmp(magic+4, DEX_OPT_MAGIC_VERS, 4) != 0) {
        ALOGE("bad opt version (0x%02x %02x %02x %02x)",
            magic[4], magic[5], magic[6], magic[7]);
    goto bail;
    }

    pDexFile->pOptHeader = (constDexOptHeader*) data;
    ALOGV("Good opt header, DEX offset is %d, flags=0x%02x",
        pDexFile->pOptHeader->dexOffset, pDexFile->pOptHeader->flags);

/* parse the optimized dex file tables */
if (!dexParseOptData(data, length, pDexFile))
    goto bail;

/* ignore the opt header and appended data from here on out */
    data += pDexFile->pOptHeader->dexOffset;
    length -= pDexFile->pOptHeader->dexOffset;
if (pDexFile->pOptHeader->dexLength > length) {
        ALOGE("File truncated? stored len=%d, rem len=%d",
            pDexFile->pOptHeader->dexLength, (int) length);
    goto bail;
    }
    length = pDexFile->pOptHeader->dexLength;
}

    dexFileSetupBasicPointers(pDexFile, data);
    pHeader = pDexFile->pHeader;

if (!dexIsValidMagic(pHeader)) {
    goto bail;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/*
 * Verify the checksum(s). This is reasonably quick, but does require
 * touching every byte in the DEX file. The base checksum changes after
 * byte-swapping and DEX optimization.
 */
if (flags & kDexParseVerifyChecksum) {
    u4 adler = dexComputeChecksum(pHeader);
    if (adler != pHeader->checksum) {
        ALOGE("ERROR: bad checksum (%08x vs %08x)",
            adler, pHeader->checksum);
    }
    if (!(flags & kDexParseContinueOnError))
        goto bail;
    } else {
        ALOGV("+++ adler32 checksum (%08x) verified", adler);
    }

    const DexOptHeader* pOptHeader = pDexFile->pOptHeader;
    if (pOptHeader != NULL) {
        adler = dexComputeOptChecksum(pOptHeader);
        if (adler != pOptHeader->checksum) {
            ALOGE("ERROR: bad opt checksum (%08x vs %08x)",
                adler, pOptHeader->checksum);
        }
        if (!(flags & kDexParseContinueOnError))
            goto bail;
        } else {
            ALOGV("+++ adler32 opt checksum (%08x) verified", adler);
        }
    }
}

/*
 * Verify the SHA-1 digest. (Normally we don't want to do this --
 * the digest is used to uniquely identify the original DEX file, and
 * can't be computed for verification after the DEX is byte-swapped
 * and optimized.)
 */
if (kVerifySignature) {
    unsigned char sha1Digest[kSHA1DigestLen];
    const int nonSum = sizeof(pHeader->magic) + sizeof(pHeader->checksum) +
        kSHA1DigestLen;

    dexComputeSHA1Digest(data + nonSum, length - nonSum, sha1Digest);
    if (memcmp(sha1Digest, pHeader->signature, kSHA1DigestLen) != 0) {
        char tmpBuf1[kSHA1DigestOutputLen];
        char tmpBuf2[kSHA1DigestOutputLen];
        ALOGE("ERROR: bad SHA1 digest (%s vs %s)",
            dexSHA1DigestToStr(sha1Digest, tmpBuf1),
            dexSHA1DigestToStr(pHeader->signature, tmpBuf2));
    }
    if (!(flags & kDexParseContinueOnError))
        goto bail;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    } else {
        ALOGV("+++ sha1 digest verified");
    }
}

if (pHeader->fileSize != length) {
    ALOGE("ERROR: stored file size (%d) != expected (%d)",
        (int) pHeader->fileSize, (int) length);
    if (!(flags & kDexParseContinueOnError))
        goto bail;
}

if (pHeader->classDefsSize == 0) {
    ALOGE("ERROR: DEX file has no classes in it, failing");
    goto bail;
}

/*
 * Success!
 */
result = 0;

bail:
if (result != 0 && pDexFile != NULL) {
    dexFileFree(pDexFile);
    pDexFile = NULL;
}
return pDexFile;
}

```

xvii      \platform\_dalvik-master\vm\DvmDex.cpp

```

/*
 * Create auxillary data structures.
 *
 * We need a 4-byte pointer for every reference to a class, method, field,
 * or string constant. Summed up over all loaded DEX files (including the
 * whoppers in the bootstrap class path), this adds up to be quite a bit
 * of native memory.
 *
 * For more traditional VMs these values could be stuffed into the loaded
 * class file constant pool area, but we don't have that luxury since our
 * classes are memory-mapped read-only.
 *
 * The DEX optimizer will remove the need for some of these (e.g. we won't
 * use the entry for virtual methods that are only called through

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* invoke-virtual-quick), creating the possibility of some space reduction
* atdexopt time.
*/

```

```

static DvmDex* allocateAuxStructures(DexFile* pDexFile)
{
    DvmDex* pDvmDex;
    const DexHeader* pHeader;
    u4stringSize, classSize, methodSize, fieldSize;

    pHeader = pDexFile->pHeader;

    stringSize = pHeader->stringIdsSize * sizeof(structStringObject*);
    classSize = pHeader->typeIdsSize * sizeof(structClassObject*);
    methodSize = pHeader->methodIdsSize * sizeof(structMethod*);
    fieldSize = pHeader->fieldIdsSize * sizeof(structField*);

    u4totalSize = sizeof(DvmDex) +
        stringSize + classSize + methodSize + fieldSize;

    u1 *blob = (u1 *)dvmAllocRegion(totalSize,
        PROT_READ | PROT_WRITE, "dalvik-aux-structure");
    if ((void *)blob == MAP_FAILED)
        return NULL;

    pDvmDex = (DvmDex*)blob;
    blob += sizeof(DvmDex);

    pDvmDex->pDexFile = pDexFile;
    pDvmDex->pHeader = pHeader;

    pDvmDex->pResStrings = (structStringObject**)blob;
    blob += stringSize;
    pDvmDex->pResClasses = (structClassObject**)blob;
    blob += classSize;
    pDvmDex->pResMethods = (structMethod**)blob;
    blob += methodSize;
    pDvmDex->pResFields = (structField**)blob;

    ALOGV("+++ DEX %p: allocateAux (%d+%d+%d+%d)*4 = %d bytes",
        pDvmDex, stringSize/4, classSize/4, methodSize/4, fieldSize/4,
        stringSize + classSize + methodSize + fieldSize);

    pDvmDex->pInterfaceCache = dvmAllocAtomicCache(DEX_INTERFACE_CACHE_SIZE);

    dvmInitMutex(&pDvmDex->modLock);

    return pDvmDex;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    \platform_dalvik-master\vm\DvmDex.h

/*
 * Some additional VM data structures that are associated with the DEX file.
 */
struct DvmDex {
/* pointer to the DexFile we're associated with */
DexFile*      pDexFile;

/* clone of pDexFile->pHeader (it's used frequently enough) */
constDexHeader* pHeader;

/* interned strings; parallel to "stringIds" */
structStringObject** pResStrings;

/* resolved classes; parallel to "typeIds" */
structClassObject** pResClasses;

/* resolved methods; parallel to "methodIds" */
structMethod**      pResMethods;

/* resolved instance fields; parallel to "fieldIds" */
/* (this holds both InstField and StaticField) */
structField**       pResFields;

/* interface method lookup cache */
structAtomicCache* pInterfaceCache;

/* shared memory region with file contents */
boolisMappedReadOnly;
MemMappingmemMap;

jobjectdex_object;

/* lock ensuring mutual exclusion during updates */
pthread_mutex_tmodLock;
};

```

xviii      Launcher2-master\src\com\android\launcher2\Launcher.java

```

/**
 * Default launcher application.
 */
public final class Launcher extends Activity
    implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks,

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    View.OnTouchListener {
    static final String TAG = "Launcher";
    static final boolean LOGD = false;

    static final boolean PROFILE_STARTUP = false;
    static final boolean DEBUG_WIDGETS = false;
    static final boolean DEBUG_STRICT_MODE = false;
    static final boolean DEBUG_RESUME_TIME = false;

    private static final int MENU_GROUP_WALLPAPER = 1;
    private static final int MENU_WALLPAPER_SETTINGS = Menu.FIRST + 1;
    private static final int MENU_MANAGE_APPS = MENU_WALLPAPER_SETTINGS + 1;
    private static final int MENU_SYSTEM_SETTINGS = MENU_MANAGE_APPS + 1;
    private static final int MENU_HELP = MENU_SYSTEM_SETTINGS + 1;

    private static final int REQUEST_CREATE_SHORTCUT = 1;
    private static final int REQUEST_CREATE_APPWIDGET = 5;
    private static final int REQUEST_PICK_APPLICATION = 6;
    private static final int REQUEST_PICK_SHORTCUT = 7;
    private static final int REQUEST_PICK_APPWIDGET = 9;
    private static final int REQUEST_PICK_WALLPAPER = 10;

    private static final int REQUEST_BIND_APPWIDGET = 11;

    static final String EXTRA_SHORTCUT_DUPLICATE = "duplicate";

    static final int SCREEN_COUNT = 5;
    static final int DEFAULT_SCREEN = 2;

    private static final String PREFERENCES = "launcher.preferences";
    // To turn on these properties, type
    // adb shell setprop log.tag.PROPERTY_NAME [VERBOSE | SUPPRESS]
    static final String FORCE_ENABLE_ROTATION_PROPERTY = "launcher_force_rotate";
    static final String DUMP_STATE_PROPERTY = "launcher_dump_state";

    // The Intent extra that defines whether to ignore the launch animation
    static final String INTENT_EXTRA_IGNORE_LAUNCH_ANIMATION =
        "com.android.launcher.intent.extra.shortcut.IGNORE_LAUNCH_ANIMATION";

    // Type: int
    private static final String RUNTIME_STATE_CURRENT_SCREEN =
"launcher.current_screen";
    // Type: int
    private static final String RUNTIME_STATE = "launcher.state";
    // Type: int
    private static final String RUNTIME_STATE_PENDING_ADD_CONTAINER =
"launcher.add_container";
    // Type: int
    private static final String RUNTIME_STATE_PENDING_ADD_SCREEN =
"launcher.add_screen";

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

// Type: int
private static final String RUNTIME_STATE_PENDING_ADD_CELL_X =
"launcher.add_cell_x";
// Type: int
private static final String RUNTIME_STATE_PENDING_ADD_CELL_Y =
"launcher.add_cell_y";
// Type: boolean
private static final String RUNTIME_STATE_PENDING_FOLDER_RENAME =
"launcher.rename_folder";
// Type: long
private static final String RUNTIME_STATE_PENDING_FOLDER_RENAME_ID =
"launcher.rename_folder_id";
// Type: int
private static final String RUNTIME_STATE_PENDING_ADD_SPAN_X =
"launcher.add_span_x";
// Type: int
private static final String RUNTIME_STATE_PENDING_ADD_SPAN_Y =
"launcher.add_span_y";
// Type: parcelable
private static final String RUNTIME_STATE_PENDING_ADD_WIDGET_INFO =
"launcher.add_widget_info";
// Type: parcelable
private static final String RUNTIME_STATE_PENDING_ADD_WIDGET_ID =
"launcher.add_widget_id";

private static final String TOOLBAR_ICON_METADATA_NAME =
"com.android.launcher.toolbar_icon";
private static final String TOOLBAR_SEARCH_ICON_METADATA_NAME =
"com.android.launcher.toolbar_search_icon";
private static final String TOOLBAR_VOICE_SEARCH_ICON_METADATA_NAME =
"com.android.launcher.toolbar_voice_search_icon";

/** The different states that Launcher can be in. */
private enum State { NONE, WORKSPACE, APPS_CUSTOMIZE,
APPS_CUSTOMIZE_SPRING_LOADED };
private State mState = State.WORKSPACE;
private AnimatorSet mStateAnimation;
private AnimatorSet mDividerAnimator;

static final int APPWIDGET_HOST_ID = 1024;
private static final int EXIT_SPRINGLOADED_MODE_SHORT_TIMEOUT = 300;
private static final int EXIT_SPRINGLOADED_MODE_LONG_TIMEOUT = 600;
private static final int SHOW_CLING_DURATION = 550;
private static final int DISMISS_CLING_DURATION = 250;

private static final Object sLock = new Object();
private static int sScreen = DEFAULT_SCREEN;

// How long to wait before the new-shortcut animation automatically pans the workspace
private static int NEW_APPS_ANIMATION_INACTIVE_TIMEOUT_SECONDS = 10;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

private final BroadcastReceiver mCloseSystemDialogsReceiver
    = new CloseSystemDialogsIntentReceiver();
private final ContentObserver mWidgetObserver = new AppWidgetResetObserver();

private LayoutInflater mInflater;

private Workspace mWorkspace;
private View mQsbDivider;
private View mDockDivider;
private View mLauncherView;
private DragLayer mDragLayer;
private DragController mDragController;

private AppWidgetManager mAppWidgetManager;
private LauncherAppWidgetHost mAppWidgetHost;

private ItemInfo mPendingAddInfo = new ItemInfo();
private AppWidgetProviderInfo mPendingAddWidgetInfo;
private int mPendingAddWidgetId = -1;

private int[] mTmpAddItemCellCoordinates = new int[2];

private FolderInfo mFolderInfo;

private Hotseat mHotseat;
private View mAllAppsButton;

private SearchDropTargetBar mSearchDropTargetBar;
private AppsCustomizeTabHost mAppsCustomizeTabHost;
private AppsCustomizePagedView mAppsCustomizeContent;
private boolean mAutoAdvanceRunning = false;

private Bundle mSavedState;
// We set the state in both onCreate and then onNewIntent in some cases, which causes both
// scroll issues (because the workspace may not have been measured yet) and extra work.
// Instead, just save the state that we need to restore Launcher to, and commit it in
onResume.
private State mOnResumeState = State.NONE;

private SpannableStringBuilder mDefaultKeySsb = null;

private boolean mWorkspaceLoading = true;

private boolean mPaused = true;
private boolean mRestoring;
private boolean mWaitingForResult;
private boolean mOnResumeNeedsLoad;

private ArrayList<Runnable> mOnResumeCallbacks = new ArrayList<Runnable>();

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

// Keep track of whether the user has left launcher
private static boolean sPausedFromUserAction = false;

private Bundle mSavedInstanceState;

private LauncherModel mModel;
private IconCache mIconCache;
private boolean mUserPresent = true;
private boolean mVisible = false;
private boolean mAttached = false;

private static LocaleConfiguration sLocaleConfiguration = null;

private static HashMap<Long, FolderInfo> sFolders = new HashMap<Long, FolderInfo>();

private Intent mAppMarketIntent = null;

// Related to the auto-advancing of widgets
private final int ADVANCE_MSG = 1;
private final int mAdvanceInterval = 20000;
private final int mAdvanceStagger = 250;
private long mAutoAdvanceSentTime;
private long mAutoAdvanceTimeLeft = -1;
private HashMap<View, AppWidgetProviderInfo> mWidgetsToAdvance =
    new HashMap<View, AppWidgetProviderInfo>();

// Determines how long to wait after a rotation before restoring the screen orientation to
// match the sensor state.
private final int mRestoreScreenOrientationDelay = 500;

// External icons saved in case of resource changes, orientation, etc.
private static Drawable.ConstantState[] sGlobalSearchIcon = new
Drawable.ConstantState[2];
private static Drawable.ConstantState[] sVoiceSearchIcon = new Drawable.ConstantState[2];
private static Drawable.ConstantState[] sAppMarketIcon = new Drawable.ConstantState[2];

private Drawable mWorkspaceBackgroundDrawable;

private final ArrayList<Integer> mSynchronouslyBoundPages = new ArrayList<Integer>();

static final ArrayList<String> sDumpLogs = new ArrayList<String>();

// We only want to get the SharedPreferences once since it does an FS stat each time we get
// it from the context.
private SharedPreferences mSharedPrefs;

// Holds the page that we need to animate to, and the icon views that we need to animate up
// when we scroll to that page on resume.
private int mNewShortcutAnimatePage = -1;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

private ArrayList<View> mNewShortcutAnimateViews = new ArrayList<View>();
private ImageView mFolderIconImageView;
private Bitmap mFolderIconBitmap;
private Canvas mFolderIconCanvas;
private Rect mRectForFolderAnimation = new Rect();

private BubbleTextView mWaitingForResume;

private HideFromAccessibilityHelper mHideFromAccessibilityHelper
    = new HideFromAccessibilityHelper();

private Runnable mBuildLayersRunnable = new Runnable() {
    public void run() {
        if (mWorkspace != null) {
            mWorkspace.buildPageHardwareLayers();
        }
    }
};

private static ArrayList<PendingAddArguments> sPendingAddList
    = new ArrayList<PendingAddArguments>();

private static boolean sForceEnableRotation =
isPropertyEnabled(FORCE_ENABLE_ROTATION_PROPERTY);

private static class PendingAddArguments {
    int requestCode;
    Intent intent;
    long container;
    int screen;
    int cellX;
    int cellY;
}

private static boolean isPropertyEnabled(String propertyName) {
    return Log.isLoggable(propertyName, Log.VERBOSE);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    if (DEBUG_STRICT_MODE) {
        StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
            .detectDiskReads()
            .detectDiskWrites()
            .detectNetwork() // or .detectAll() for all detectable problems
            .penaltyLog()
            .build());
        StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
            .detectLeakedSqlLiteObjects()
            .detectLeakedClosableObjects()

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        .penaltyLog()
        .penaltyDeath()
        .build());
    }

    super.onCreate(savedInstanceState);
    LauncherApplication app = ((LauncherApplication)getApplication());
    mSharedPreferences = getSharedPreferences(LauncherApplication.getSharedPreferencesKey(),
        Context.MODE_PRIVATE);
    mModel = app.setLauncher(this);
    mIconCache = app.getIconCache();
    mDragController = new DragController(this);
    mInflater = getLayoutInflater();

    mAppWidgetManager = AppWidgetManager.getInstance(this);
    mAppWidgetHost = new LauncherAppWidgetHost(this, APPWIDGET_HOST_ID);
    mAppWidgetHost.startListening();

    // If we are getting an onCreate, we can actually preempt onResume and unset mPaused
here,
    // this also ensures that any synchronous binding below doesn't re-trigger another
    // LauncherModel load.
    mPaused = false;

    if (PROFILE_STARTUP) {
        android.os.Debug.startMethodTracing(
            Environment.getExternalStorageDirectory() + "/launcher");
    }

    checkForLocaleChange();
    setContentView(R.layout.launcher);
    setupViews();
    showFirstRunWorkspaceCling();

    registerContentObservers();

    lockAllApps();

    mSavedState = savedInstanceState;
    restoreState(mSavedState);

    // Update customization drawer _after_ restoring the states
    if (mAppsCustomizeContent != null) {
        mAppsCustomizeContent.onPackagesUpdated(
            LauncherModel.getSortedWidgetsAndShortcuts(this));
    }

    if (PROFILE_STARTUP) {
        android.os.Debug.stopMethodTracing();
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

if (!mRestoring) {
    if (sPausedFromUserAction) {
        // If the user leaves launcher, then we should just load items asynchronously when
        // they return.
        mModel.startLoader(true, -1);
    } else {
        // We only load the page synchronously if the user rotates (or triggers a
        // configuration change) while launcher is in the foreground
        mModel.startLoader(true, mWorkspace.getCurrentPage());
    }
}

if (!mModel.isAllAppsLoaded()) {
    ViewGroup appsCustomizeContentParent = (ViewGroup)
mAppsCustomizeContent.getParent();
    mInflater.inflate(R.layout.apps_customize_progressbar, appsCustomizeContentParent);
}

// For handling default keys
mDefaultKeySsb = new SpannableStringBuilder();
Selection.setSelection(mDefaultKeySsb, 0);

IntentFilter filter = new IntentFilter(Intent.ACTION_CLOSE_SYSTEM_DIALOGS);
registerReceiver(mCloseSystemDialogsReceiver, filter);

updateGlobalIcons();

// On large interfaces, we want the screen to auto-rotate based on the current orientation
unlockScreenOrientation(true);
}

protected void onUserLeaveHint() {
    super.onUserLeaveHint();
    sPausedFromUserAction = true;
}

private void updateGlobalIcons() {
    boolean searchVisible = false;
    boolean voiceVisible = false;
    // If we have a saved version of these external icons, we load them up immediately
    int coi = getCurrentOrientationIndexForGlobalIcons();
    if (sGlobalSearchIcon[coi] == null || sVoiceSearchIcon[coi] == null ||
        sAppMarketIcon[coi] == null) {
        updateAppMarketIcon();
        searchVisible = updateGlobalSearchIcon();
        voiceVisible = updateVoiceSearchIcon(searchVisible);
    }
    if (sGlobalSearchIcon[coi] != null) {
        updateGlobalSearchIcon(sGlobalSearchIcon[coi]);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        searchVisible = true;
    }
    if (sVoiceSearchIcon[coi] != null) {
        updateVoiceSearchIcon(sVoiceSearchIcon[coi]);
        voiceVisible = true;
    }
    if (sAppMarketIcon[coi] != null) {
        updateAppMarketIcon(sAppMarketIcon[coi]);
    }
    if (mSearchDropTargetBar != null) {
        mSearchDropTargetBar.onSearchPackagesChanged(searchVisible, voiceVisible);
    }
}

private void checkForLocaleChange() {
    if (sLocaleConfiguration == null) {
        new AsyncTask<Void, Void, LocaleConfiguration>() {
            @Override
            protected LocaleConfiguration doInBackground(Void... unused) {
                LocaleConfiguration localeConfiguration = new LocaleConfiguration();
                readConfiguration(Launcher.this, localeConfiguration);
                return localeConfiguration;
            }

            @Override
            protected void onPostExecute(LocaleConfiguration result) {
                sLocaleConfiguration = result;
                checkForLocaleChange(); // recursive, but now with a locale configuration
            }
        }.execute();
        return;
    }

    final Configuration configuration = getResources().getConfiguration();

    final String previousLocale = sLocaleConfiguration.locale;
    final String locale = configuration.locale.toString();

    final int previousMcc = sLocaleConfiguration.mcc;
    final int mcc = configuration.mcc;

    final int previousMnc = sLocaleConfiguration.mnc;
    final int mnc = configuration.mnc;

    boolean localeChanged = !locale.equals(previousLocale) || mcc != previousMcc || mnc !=
previousMnc;

    if (localeChanged) {
        sLocaleConfiguration.locale = locale;
        sLocaleConfiguration.mcc = mcc;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        sLocaleConfiguration.mnc = mnc;

        mlconCache.flush();

        final LocaleConfiguration localeConfiguration = sLocaleConfiguration;
        new Thread("WriteLocaleConfiguration") {
            @Override
            public void run() {
                writeConfiguration(Launcher.this, localeConfiguration);
            }
        }.start();
    }
}

private static class LocaleConfiguration {
    public String locale;
    public int mcc = -1;
    public int mnc = -1;
}

private static void readConfiguration(Context context, LocaleConfiguration configuration) {
    DataInputStream in = null;
    try {
        in = new DataInputStream(context.openFileInput(PREFERENCES));
        configuration.locale = in.readUTF();
        configuration.mcc = in.readInt();
        configuration.mnc = in.readInt();
    } catch (FileNotFoundException e) {
        // Ignore
    } catch (IOException e) {
        // Ignore
    } finally {
        if (in != null) {
            try {
                in.close();
            } catch (IOException e) {
                // Ignore
            }
        }
    }
}

private static void writeConfiguration(Context context, LocaleConfiguration configuration) {
    DataOutputStream out = null;
    try {
        out = new DataOutputStream(context.openFileOutput(PREFERENCES,
MODE_PRIVATE));
        out.writeUTF(configuration.locale);
        out.writeInt(configuration.mcc);
        out.writeInt(configuration.mnc);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        out.flush();
    } catch (FileNotFoundException e) {
        // Ignore
    } catch (IOException e) {
        //noinspection ResultOfMethodCallIgnored
        context.getFilePath(PREFERENCES).delete();
    } finally {
        if (out != null) {
            try {
                out.close();
            } catch (IOException e) {
                // Ignore
            }
        }
    }
}

public DragLayer getDragLayer() {
    return mDragLayer;
}

boolean isDraggingEnabled() {
    // We prevent dragging when we are loading the workspace as it is possible to pick up a
view
    // that is subsequently removed from the workspace in startBinding().
    return !mModel.isLoadingWorkspace();
}

static int getScreen() {
    synchronized (sLock) {
        return sScreen;
    }
}

static void setScreen(int screen) {
    synchronized (sLock) {
        sScreen = screen;
    }
}

/**
 * Returns whether we should delay spring loaded mode -- for shortcuts and widgets that
have
 * a configuration step, this allows the proper animations to run after other transitions.
 */
private boolean completeAdd(PendingAddArguments args) {
    boolean result = false;
    switch (args.requestCode) {
        case REQUEST_PICK_APPLICATION:
            completeAddApplication(args.intent, args.container, args.screen, args.cellX,

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        args.cellY);
        break;
    case REQUEST_PICK_SHORTCUT:
        processShortcut(args.intent);
        break;
    case REQUEST_CREATE_SHORTCUT:
        completeAddShortcut(args.intent, args.container, args.screen, args.cellX,
            args.cellY);
        result = true;
        break;
    case REQUEST_CREATE_APPWIDGET:
        int appWidgetId =
args.intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, -1);
        completeAddAppWidget(appWidgetId, args.container, args.screen, null, null);
        result = true;
        break;
    case REQUEST_PICK_WALLPAPER:
        // We just wanted the activity result here so we can clear mWaitingForResult
        break;
}
// Before adding this resetAddInfo(), after a shortcut was added to a workspace screen,
// if you turned the screen off and then back while in All Apps, Launcher would not
// return to the workspace. Clearing mAddInfo.container here fixes this issue
resetAddInfo();
return result;
}

@Override
protected void onActivityResult(
    final int requestCode, final int resultCode, final Intent data) {

    int pendingAddWidgetId = mPendingAddWidgetId;
    mPendingAddWidgetId = -1;

    if (requestCode == REQUEST_BIND_APPWIDGET) {
        int appWidgetId = data != null ?
            data.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, -1) : -1;
        if (resultCode == RESULT_CANCELED) {
            completeTwoStageWidgetDrop(RESULT_CANCELED, appWidgetId);
        } else if (resultCode == RESULT_OK) {
            addAppWidgetImpl(appWidgetId, mPendingAddInfo, null, mPendingAddWidgetInfo);
        }
        return;
    }
    boolean delayExitSpringLoadedMode = false;
    boolean isWidgetDrop = (requestCode == REQUEST_PICK_APPWIDGET ||
        requestCode == REQUEST_CREATE_APPWIDGET);
    mWaitingForResult = false;

    // We have special handling for widgets

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        if (isWidgetDrop) {
            final int appWidgetId;
            int widgetId = data != null ?
data.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, -1)
                : -1;
            if (widgetId < 0) {
                appWidgetId = pendingAddWidgetId;
            } else {
                appWidgetId = widgetId;
            }

            if (appWidgetId < 0) {
                Log.e(TAG, "Error: appWidgetId (EXTRA_APPWIDGET_ID) was not returned from
the \\" +
                    "widget configuration activity.");
                completeTwoStageWidgetDrop(RESULT_CANCELED, appWidgetId);
            } else {
                completeTwoStageWidgetDrop(resultCode, appWidgetId);
            }
            return;
        }

// The pattern used here is that a user PICKs a specific application,
// which, depending on the target, might need to CREATE the actual target.

// For example, the user would PICK_SHORTCUT for "Music playlist", and we
// launch over to the Music app to actually CREATE_SHORTCUT.
if (resultCode == RESULT_OK && mPendingAddInfo.container != ItemInfo.NO_ID) {
    final PendingAddArguments args = new PendingAddArguments();
    args.requestCode = requestCode;
    args.intent = data;
    args.container = mPendingAddInfo.container;
    args.screen = mPendingAddInfo.screen;
    args.cellX = mPendingAddInfo.cellX;
    args.cellY = mPendingAddInfo.cellY;
    if (isWorkspaceLocked()) {
        sPendingAddList.add(args);
    } else {
        delayExitSpringLoadedMode = completeAdd(args);
    }
}
mDragLayer.clearAnimatedView();
// Exit spring loaded mode if necessary after cancelling the configuration of a widget
exitSpringLoadedDragModeDelayed((resultCode != RESULT_CANCELED),
delayExitSpringLoadedMode,
    null);
}

private void completeTwoStageWidgetDrop(final int resultCode, final int appWidgetId) {
    CellLayout cellLayout =

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        (CellLayout) mWorkspace.getChildAt(mPendingAddInfo.screen);
        Runnable onCompleteRunnable = null;
        int animationType = 0;

        AppWidgetHostView boundWidget = null;
        if (resultCode == RESULT_OK) {
            animationType =
Workspace.COMPLETE_TWO_STAGE_WIDGET_DROP_ANIMATION;
            final AppWidgetHostView layout = mAppWidgetHost.createView(this, appWidgetId,
                mPendingAddWidgetInfo);
            boundWidget = layout;
            onCompleteRunnable = new Runnable() {
                @Override
                public void run() {
                    completeAddAppWidget(appWidgetId, mPendingAddInfo.container,
                        mPendingAddInfo.screen, layout, null);
                    exitSpringLoadedDragModeDelayed((resultCode != RESULT_CANCELED), false,
                        null);
                }
            };
        } else if (resultCode == RESULT_CANCELED) {
            mAppWidgetHost.deleteAppWidgetId(appWidgetId);
            animationType = Workspace.CANCEL_TWO_STAGE_WIDGET_DROP_ANIMATION;
            onCompleteRunnable = new Runnable() {
                @Override
                public void run() {
                    exitSpringLoadedDragModeDelayed((resultCode != RESULT_CANCELED), false,
                        null);
                }
            };
        }
        if (mDragLayer.getAnimatedView() != null) {
            mWorkspace.animateWidgetDrop(mPendingAddInfo, cellLayout,
                (DragView) mDragLayer.getAnimatedView(), onCompleteRunnable,
                animationType, boundWidget, true);
        } else {
            // The animated view may be null in the case of a rotation during widget configuration
            onCompleteRunnable.run();
        }
    }

    @Override
    protected void onStop() {
        super.onStop();
        FirstFrameAnimatorHelper.setIsVisible(false);
    }

    @Override
    protected void onStart() {
        super.onStart();

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    FirstFrameAnimatorHelper.setIsVisible(true);
}

@Override
protected void onResume() {
    long startTime = 0;
    if (DEBUG_RESUME_TIME) {
        startTime = System.currentTimeMillis();
    }
    super.onResume();

    // Restore the previous launcher state
    if (mOnResumeState == State.WORKSPACE) {
        showWorkspace(false);
    } else if (mOnResumeState == State.APPS_CUSTOMIZE) {
        showAllApps(false);
    }
    mOnResumeState = State.NONE;

    // Background was set to gradient in onPause(), restore to black if in all apps.
    setWorkspaceBackground(mState == State.WORKSPACE);

    // Process any items that were added while Launcher was away
    InstallShortcutReceiver.flushInstallQueue(this);

    mPaused = false;
    sPausedFromUserAction = false;
    if (mRestoring || mOnResumeNeedsLoad) {
        mWorkspaceLoading = true;
        mModel.startLoader(true, -1);
        mRestoring = false;
        mOnResumeNeedsLoad = false;
    }
    if (mOnResumeCallbacks.size() > 0) {
        // We might have postponed some bind calls until onResume (see waitUntilResume) --
        // execute them here
        long startTimeCallbacks = 0;
        if (DEBUG_RESUME_TIME) {
            startTimeCallbacks = System.currentTimeMillis();
        }

        if (mAppsCustomizeContent != null) {
            mAppsCustomizeContent.setBulkBind(true);
        }
        for (int i = 0; i < mOnResumeCallbacks.size(); i++) {
            mOnResumeCallbacks.get(i).run();
        }
        if (mAppsCustomizeContent != null) {
            mAppsCustomizeContent.setBulkBind(false);
        }
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        mOnResumeCallbacks.clear();
        if (DEBUG_RESUME_TIME) {
            Log.d(TAG, "Time spent processing callbacks in onResume: " +
                (System.currentTimeMillis() - startTimeCallbacks));
        }
    }

    // Reset the pressed state of icons that were locked in the press state while activities
    // were launching
    if (mWaitingForResume != null) {
        // Resets the previous workspace icon press state
        mWaitingForResume.setStayPressed(false);
    }
    if (mAppsCustomizeContent != null) {
        // Resets the previous all apps icon press state
        mAppsCustomizeContent.resetDrawableState();
    }
    // It is possible that widgets can receive updates while launcher is not in the foreground.
    // Consequently, the widgets will be inflated in the orientation of the foreground activity
    // (framework issue). On resuming, we ensure that any widgets are inflated for the current
    // orientation.
    getWorkspace().reinflateWidgetsIfNecessary();

    // Again, as with the above scenario, it's possible that one or more of the global icons
    // were updated in the wrong orientation.
    updateGlobalIcons();
    if (DEBUG_RESUME_TIME) {
        Log.d(TAG, "Time spent in onResume: " + (System.currentTimeMillis() - startTime));
    }
}

@Override
protected void onPause() {
    // NOTE: We want all transitions from launcher to act as if the wallpaper were enabled
    // to be consistent. So re-enable the flag here, and we will re-disable it as necessary
    // when Launcher resumes and we are still in AllApps.
    updateWallpaperVisibility(true);

    super.onPause();
    mPaused = true;
    mDragController.cancelDrag();
    mDragController.resetLastGestureUpTime();
}

@Override
public Object onRetainNonConfigurationInstance() {
    // Flag the loader to stop early before switching
    mModel.stopLoader();
    if (mAppsCustomizeContent != null) {
        mAppsCustomizeContent.surrender();
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }
    return Boolean.TRUE;
}

// We can't hide the IME if it was forced open. So don't bother
/*
@Override
public void onWindowFocusChanged(boolean hasFocus) {
    super.onWindowFocusChanged(hasFocus);

    if (hasFocus) {
        final InputMethodManager inputManager = (InputMethodManager)
            getSystemService(Context.INPUT_METHOD_SERVICE);
        WindowManager.LayoutParams lp = getWindow().getAttributes();
        inputManager.hideSoftInputFromWindow(lp.token, 0, new
android.os.ResultReceiver(new
        android.os.Handler()) {
            protected void onReceiveResult(int resultCode, Bundle resultData) {
                Log.d(TAG, "ResultReceiver got resultCode=" + resultCode);
            }
        });
        Log.d(TAG, "called hideSoftInputFromWindow from onWindowFocusChanged");
    }
}
*/

private boolean acceptFilter() {
    final InputMethodManager inputManager = (InputMethodManager)
        getSystemService(Context.INPUT_METHOD_SERVICE);
    return !inputManager.isFullscreenMode();
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    final int uniChar = event.getUnicodeChar();
    final boolean handled = super.onKeyDown(keyCode, event);
    final boolean isKeyNotWhitespace = uniChar > 0 && !Character.isWhitespace(uniChar);
    if (!handled && acceptFilter() && isKeyNotWhitespace) {
        boolean gotKey = TextKeyListener.getInstance().onKeyDown(mWorkspace,
mDefaultKeySsb,
        keyCode, event);
        if (gotKey && mDefaultKeySsb != null && mDefaultKeySsb.length() > 0) {
            // something usable has been typed - start a search
            // the typed text will be retrieved and cleared by
            // showSearchDialog()
            // If there are multiple keystrokes before the search dialog takes focus,
            // onSearchRequested() will be called for every keystroke,
            // but it is idempotent, so it's fine.
            return onSearchRequested();
        }
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    // Eat the long press event so the keyboard doesn't come up.
    if (keyCode == KeyEvent.KEYCODE_MENU && event.isLongPress()) {
        return true;
    }

    return handled;
}

private String getTypedText() {
    return mDefaultKeySsb.toString();
}

private void clearTypedText() {
    mDefaultKeySsb.clear();
    mDefaultKeySsb.clearSpans();
    Selection.setSelection(mDefaultKeySsb, 0);
}

/**
 * Given the integer (ordinal) value of a State enum instance, convert it to a variable of type
 * State
 */
private static State intToState(int stateOrdinal) {
    State state = State.WORKSPACE;
    final State[] stateValues = State.values();
    for (int i = 0; i < stateValues.length; i++) {
        if (stateValues[i].ordinal() == stateOrdinal) {
            state = stateValues[i];
            break;
        }
    }
    return state;
}

/**
 * Restores the previous state, if it exists.
 *
 * @param savedInstanceState The previous state.
 */
private void restoreState(Bundle savedInstanceState) {
    if (savedInstanceState == null) {
        return;
    }

    State state = intToState(savedInstanceState.getInt(RUNTIME_STATE,
        State.WORKSPACE.ordinal()));
    if (state == State.APPS_CUSTOMIZE) {
        mOnResumeState = State.APPS_CUSTOMIZE;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    int currentScreen = savedInstanceState.getInt(RUNTIME_STATE_CURRENT_SCREEN, -1);
    if (currentScreen > -1) {
        mWorkspace.setCurrentPage(currentScreen);
    }

    final long pendingAddContainer =
savedState.getLong(RUNTIME_STATE_PENDING_ADD_CONTAINER, -1);
    final int pendingAddScreen =
savedState.getInt(RUNTIME_STATE_PENDING_ADD_SCREEN, -1);

    if (pendingAddContainer != ItemInfo.NO_ID && pendingAddScreen > -1) {
        mPendingAddInfo.container = pendingAddContainer;
        mPendingAddInfo.screen = pendingAddScreen;
        mPendingAddInfo.cellX =
savedState.getInt(RUNTIME_STATE_PENDING_ADD_CELL_X);
        mPendingAddInfo.cellY =
savedState.getInt(RUNTIME_STATE_PENDING_ADD_CELL_Y);
        mPendingAddInfo.spanX =
savedState.getInt(RUNTIME_STATE_PENDING_ADD_SPAN_X);
        mPendingAddInfo.spanY =
savedState.getInt(RUNTIME_STATE_PENDING_ADD_SPAN_Y);
        mPendingAddWidgetInfo =
savedState.getParcelable(RUNTIME_STATE_PENDING_ADD_WIDGET_INFO);
        mPendingAddWidgetId =
savedState.getInt(RUNTIME_STATE_PENDING_ADD_WIDGET_ID);
        mWaitingForResult = true;
        mRestoring = true;
    }

    boolean renameFolder =
savedState.getBoolean(RUNTIME_STATE_PENDING_FOLDER_RENAME, false);
    if (renameFolder) {
        long id = savedInstanceState.getLong(RUNTIME_STATE_PENDING_FOLDER_RENAME_ID);
        mFolderInfo = mModel.getFolderById(this, sFolders, id);
        mRestoring = true;
    }

    // Restore the AppsCustomize tab
    if (mAppsCustomizeTabHost != null) {
        String curTab = savedInstanceState.getString("apps_customize_currentTab");
        if (curTab != null) {
            mAppsCustomizeTabHost.setContentTypeImmediate(
                mAppsCustomizeTabHost.getContentTypeForTabTag(curTab));
            mAppsCustomizeContent.loadAssociatedPages(
                mAppsCustomizeContent.getCurrentPage());
        }
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        int currentIndex = savedInstanceState.getInt("apps_customize_currentIndex");
        mAppsCustomizeContent.restorePageForIndex(currentIndex);
    }
}

/**
 * Finds all the views we need and configure them properly.
 */
private void setupViews() {
    final DragController dragController = mDragController;

    mLauncherView = findViewById(R.id.launcher);
    mDragLayer = (DragLayer) findViewById(R.id.drag_layer);
    mWorkspace = (Workspace) mDragLayer.findViewById(R.id.workspace);
    mQsbDivider = findViewById(R.id.qsb_divider);
    mDockDivider = findViewById(R.id.dock_divider);

    mLauncherView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
    mWorkspace.setBackgroundDrawable =
    getResources().getDrawable(R.drawable.workspace_bg);

    // Setup the drag layer
    mDragLayer.setup(this, dragController);

    // Setup the hotseat
    mHotseat = (Hotseat) findViewById(R.id.hotseat);
    if (mHotseat != null) {
        mHotseat.setup(this);
    }

    // Setup the workspace
    mWorkspace.setHapticFeedbackEnabled(false);
    mWorkspace.setOnLongClickListener(this);
    mWorkspace.setup(dragController);
    dragController.addDragListener(mWorkspace);

    // Get the search/delete bar
    mSearchDropTargetBar = (SearchDropTargetBar)
    mDragLayer.findViewById(R.id.qsb_bar);

    // Setup AppsCustomize
    mAppsCustomizeTabHost = (AppsCustomizeTabHost)
    findViewById(R.id.apps_customize_pane);
    mAppsCustomizeContent = (AppsCustomizePagedView)
        mAppsCustomizeTabHost.findViewById(R.id.apps_customize_pane_content);
    mAppsCustomizeContent.setup(this, dragController);

    // Setup the drag controller (drop targets have to be added in reverse order in priority)

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        dragController.setDragScoller(mWorkspace);
        dragController.setScrollView(mDragLayer);
        dragController.setMoveTarget(mWorkspace);
        dragController.addDropTarget(mWorkspace);
        if (mSearchDropTargetBar != null) {
            mSearchDropTargetBar.setup(this, dragController);
        }
    }

    /**
     * Creates a view representing a shortcut.
     *
     * @param info The data structure describing the shortcut.
     *
     * @return A View inflated from R.layout.application.
     */
    View createShortcut(ShortcutInfo info) {
        return createShortcut(R.layout.application,
            (ViewGroup) mWorkspace.getChildAt(mWorkspace.getCurrentPage()), info);
    }

    /**
     * Creates a view representing a shortcut inflated from the specified resource.
     *
     * @param layoutResId The id of the XML layout used to create the shortcut.
     * @param parent The group the shortcut belongs to.
     * @param info The data structure describing the shortcut.
     *
     * @return A View inflated from layoutResId.
     */
    View createShortcut(int layoutResId, ViewGroup parent, ShortcutInfo info) {
        BubbleTextView favorite = (BubbleTextView) mInflater.inflate(layoutResId, parent, false);
        favorite.applyFromShortcutInfo(info, mIconCache);
        favorite.setOnClickListener(this);
        return favorite;
    }

    /**
     * Add an application shortcut to the workspace.
     *
     * @param data The intent describing the application.
     * @param cellInfo The position on screen where to create the shortcut.
     */
    void completeAddApplication(Intent data, long container, int screen, int cellX, int cellY) {
        final int[] cellXY = mTmpAddItemCellCoordinates;
        final CellLayout layout = getCellLayout(container, screen);

        // First we check if we already know the exact location where we want to add this item.
        if (cellX >= 0 && cellY >= 0) {
            cellXY[0] = cellX;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        cellXY[1] = cellY;
    } else if (!layout.findCellForSpan(cellXY, 1, 1)) {
        showOutOfSpaceMessage(isHotseatLayout(layout));
        return;
    }

    final ShortcutInfo info = mModel.getShortcutInfo(getPackageManager(), data, this);

    if (info != null) {
        info.setActivity(data.getComponent(), Intent.FLAG_ACTIVITY_NEW_TASK |
            Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED);
        info.container = ItemInfo.NO_ID;
        mWorkspace.addApplicationShortcut(info, layout, container, screen, cellXY[0], cellXY[1],
            isWorkspaceLocked(), cellX, cellY);
    } else {
        Log.e(TAG, "Couldn't find ActivityInfo for selected application: " + data);
    }
}

/**
 * Add a shortcut to the workspace.
 *
 * @param data The intent describing the shortcut.
 * @param cellInfo The position on screen where to create the shortcut.
 */
private void completeAddShortcut(Intent data, long container, int screen, int cellX,
    int cellY) {
    int[] cellXY = mTmpAddItemCellCoordinates;
    int[] touchXY = mPendingAddInfo.dropPos;
    CellLayout layout = getCellLayout(container, screen);

    boolean foundCellSpan = false;

    ShortcutInfo info = mModel.infoFromShortcutIntent(this, data, null);
    if (info == null) {
        return;
    }
    final View view = createShortcut(info);

    // First we check if we already know the exact location where we want to add this item.
    if (cellX >= 0 && cellY >= 0) {
        cellXY[0] = cellX;
        cellXY[1] = cellY;
        foundCellSpan = true;

        // If appropriate, either create a folder or add to an existing folder
        if (mWorkspace.createUserFolderIfNecessary(view, container, layout, cellXY, 0,
            true, null, null)) {
            return;
        }
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        DragObject dragObject = new DragObject();
        dragObject.dragInfo = info;
        if (mWorkspace.addToExistingFolderIfNecessary(view, layout, cellXY, 0, dragObject,
            true)) {
            return;
        }
    } else if (touchXY != null) {
        // when dragging and dropping, just find the closest free spot
        int[] result = layout.findNearestVacantArea(touchXY[0], touchXY[1], 1, 1, cellXY);
        foundCellSpan = (result != null);
    } else {
        foundCellSpan = layout.findCellForSpan(cellXY, 1, 1);
    }

    if (!foundCellSpan) {
        showOutOfSpaceMessage(isHotseatLayout(layout));
        return;
    }

    LauncherModel.addItemToDatabase(this, info, container, screen, cellXY[0], cellXY[1],
false);

    if (!mRestoring) {
        mWorkspace.addInScreen(view, container, screen, cellXY[0], cellXY[1], 1, 1,
            isWorkspaceLocked());
    }
}

static int[] getSpanForWidget(Context context, ComponentName component, int minWidth,
    int minHeight) {
    Rect padding = AppWidgetHostView.getDefaultPaddingForWidget(context, component,
null);
    // We want to account for the extra amount of padding that we are adding to the widget
    // to ensure that it gets the full amount of space that it has requested
    int requiredWidth = minWidth + padding.left + padding.right;
    int requiredHeight = minHeight + padding.top + padding.bottom;
    return CellLayout.rectToCell(context.getResources(), requiredWidth, requiredHeight, null);
}

static int[] getSpanForWidget(Context context, AppWidgetProviderInfo info) {
    return getSpanForWidget(context, info.provider, info.minWidth, info.minHeight);
}

static int[] getMinSpanForWidget(Context context, AppWidgetProviderInfo info) {
    return getSpanForWidget(context, info.provider, info.minResizeWidth,
info.minResizeHeight);
}

static int[] getSpanForWidget(Context context, PendingAddWidgetInfo info) {
    return getSpanForWidget(context, info.componentName, info.minWidth, info.minHeight);
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

}

static int[] getMinSpanForWidget(Context context, PendingAddWidgetInfo info) {
    return getSpanForWidget(context, info.componentName, info.minResizeWidth,
        info.minResizeHeight);
}

/**
 * Add a widget to the workspace.
 *
 * @param appWidgetId The app widget id
 * @param cellInfo The position on screen where to create the widget.
 */
private void completeAddAppWidget(final int appWidgetId, long container, int screen,
    AppWidgetHostView hostView, AppWidgetProviderInfo appWidgetInfo) {
    if (appWidgetInfo == null) {
        appWidgetInfo = mAppWidgetManager.getAppWidgetInfo(appWidgetId);
    }

    // Calculate the grid spans needed to fit this widget
    CellLayout layout = getCellLayout(container, screen);

    int[] minSpanXY = getMinSpanForWidget(this, appWidgetInfo);
    int[] spanXY = getSpanForWidget(this, appWidgetInfo);

    // Try finding open space on Launcher screen
    // We have saved the position to which the widget was dragged-- this really only matters
    // if we are placing widgets on a "spring-loaded" screen
    int[] cellXY = mTmpAddItemCellCoordinates;
    int[] touchXY = mPendingAddInfo.dropPos;
    int[] finalSpan = new int[2];
    boolean foundCellSpan = false;
    if (mPendingAddInfo.cellX >= 0 && mPendingAddInfo.cellY >= 0) {
        cellXY[0] = mPendingAddInfo.cellX;
        cellXY[1] = mPendingAddInfo.cellY;
        spanXY[0] = mPendingAddInfo.spanX;
        spanXY[1] = mPendingAddInfo.spanY;
        foundCellSpan = true;
    } else if (touchXY != null) {
        // when dragging and dropping, just find the closest free spot
        int[] result = layout.findNearestVacantArea(
            touchXY[0], touchXY[1], minSpanXY[0], minSpanXY[1], spanXY[0],
            spanXY[1], cellXY, finalSpan);
        spanXY[0] = finalSpan[0];
        spanXY[1] = finalSpan[1];
        foundCellSpan = (result != null);
    } else {
        foundCellSpan = layout.findCellForSpan(cellXY, minSpanXY[0], minSpanXY[1]);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    if (!foundCellSpan) {
        if (appWidgetId != -1) {
            // Deleting an app widget ID is a void call but writes to disk before returning
            // to the caller...
            new Thread("deleteAppWidgetId") {
                public void run() {
                    mAppWidgetHost.deleteAppWidgetId(appWidgetId);
                }
            }.start();
        }
        showOutOfSpaceMessage(isHotseatLayout(layout));
        return;
    }

    // Build Launcher-specific widget info and save to database
    LauncherAppWidgetInfo launcherInfo = new LauncherAppWidgetInfo(appWidgetId,
        appWidgetInfo.provider);
    launcherInfo.spanX = spanXY[0];
    launcherInfo.spanY = spanXY[1];
    launcherInfo.minSpanX = mPendingAddInfo.minSpanX;
    launcherInfo.minSpanY = mPendingAddInfo.minSpanY;

    LauncherModel.addItemToDatabase(this, launcherInfo,
        container, screen, cellXY[0], cellXY[1], false);

    if (!mRestoring) {
        if (hostView == null) {
            // Perform actual inflation because we're live
            launcherInfo.hostView = mAppWidgetHost.createView(this, appWidgetId,
appWidgetInfo);
            launcherInfo.hostView.setAppWidget(appWidgetId, appWidgetInfo);
        } else {
            // The AppWidgetHostView has already been inflated and instantiated
            launcherInfo.hostView = hostView;
        }

        launcherInfo.hostView.setTag(launcherInfo);
        launcherInfo.hostView.setVisibility(View.VISIBLE);
        launcherInfo.notifyWidgetSizeChanged(this);

        mWorkspace.addInScreen(launcherInfo.hostView, container, screen, cellXY[0],
cellXY[1],
            launcherInfo.spanX, launcherInfo.spanY, isWorkspaceLocked());

        addWidgetToAutoAdvanceIfNeeded(launcherInfo.hostView, appWidgetInfo);
    }
    resetAddInfo();
}

private final BroadcastReceiver mReceiver = new BroadcastReceiver() {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

@Override
public void onReceive(Context context, Intent intent) {
    final String action = intent.getAction();
    if (Intent.ACTION_SCREEN_OFF.equals(action)) {
        mUserPresent = false;
        mDragLayer.clearAllResizeFrames();
        updateRunning();

        // Reset AllApps to its initial state only if we are not in the middle of
        // processing a multi-step drop
        if (mAppsCustomizeTabHost != null && mPendingAddInfo.container ==
ItemInfo.NO_ID) {
            mAppsCustomizeTabHost.reset();
            showWorkspace(false);
        }
    } else if (Intent.ACTION_USER_PRESENT.equals(action)) {
        mUserPresent = true;
        updateRunning();
    }
}
};

@Override
public void onAttachedToWindow() {
    super.onAttachedToWindow();

    // Listen for broadcasts related to user-presence
    final IntentFilter filter = new IntentFilter();
    filter.addAction(Intent.ACTION_SCREEN_OFF);
    filter.addAction(Intent.ACTION_USER_PRESENT);
    registerReceiver(mReceiver, filter);
    FirstFrameAnimatorHelper.initializeDrawListener(getWindow().getDecorView());
    mAttached = true;
    mVisible = true;
}

@Override
public void onDetachedFromWindow() {
    super.onDetachedFromWindow();
    mVisible = false;

    if (mAttached) {
        unregisterReceiver(mReceiver);
        mAttached = false;
    }
    updateRunning();
}

public void onWindowVisibilityChanged(int visibility) {
    mVisible = visibility == View.VISIBLE;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

updateRunning();
// The following code used to be in onResume, but it turns out onResume is called when
// you're in All Apps and click home to go to the workspace. onWindowVisibilityChanged
// is a more appropriate event to handle
if (mVisible) {
    mAppsCustomizeTabHost.onWindowVisible();
    if (!mWorkspaceLoading) {
        final ViewTreeObserver observer = mWorkspace.getViewTreeObserver();
        // We want to let Launcher draw itself at least once before we force it to build
        // layers on all the workspace pages, so that transitioning to Launcher from other
        // apps is nice and speedy.
        observer.addOnDrawListener(new ViewTreeObserver.OnDrawListener() {
            private boolean mStarted = false;
            public void onDraw() {
                if (mStarted) return;
                mStarted = true;
                // We delay the layer building a bit in order to give
                // other message processing a time to run. In particular
                // this avoids a delay in hiding the IME if it was
                // currently shown, because doing that may involve
                // some communication back with the app.
                mWorkspace.postDelayed(mBuildLayersRunnable, 500);
                final ViewTreeObserver.OnDrawListener listener = this;
                mWorkspace.post(new Runnable() {
                    public void run() {
                        if (mWorkspace != null &&
                            mWorkspace.getViewTreeObserver() != null) {
                            mWorkspace.getViewTreeObserver().
                                removeOnDrawListener(listener);
                        }
                    }
                });
            }
        });
        return;
    }
}

// When Launcher comes back to foreground, a different Activity might be responsible for
// the app market intent, so refresh the icon
updateAppMarketIcon();
clearTypedText();
}

private void sendAdvanceMessage(long delay) {
    mHandler.removeMessages(ADVANCE_MSG);
    Message msg = mHandler.obtainMessage(ADVANCE_MSG);
    mHandler.sendMessageDelayed(msg, delay);
    mAutoAdvanceSentTime = System.currentTimeMillis();
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

private void updateRunning() {
    boolean autoAdvanceRunning = mVisible && mUserPresent &&
!mWidgetsToAdvance.isEmpty();
    if (autoAdvanceRunning != mAutoAdvanceRunning) {
        mAutoAdvanceRunning = autoAdvanceRunning;
        if (autoAdvanceRunning) {
            long delay = mAutoAdvanceTimeLeft == -1 ? mAdvanceInterval :
mAutoAdvanceTimeLeft;
            sendAdvanceMessage(delay);
        } else {
            if (!mWidgetsToAdvance.isEmpty()) {
                mAutoAdvanceTimeLeft = Math.max(0, mAdvanceInterval -
                (System.currentTimeMillis() - mAutoAdvanceSentTime));
            }
            mHandler.removeMessages(ADVANCE_MSG);
            mHandler.removeMessages(0); // Remove messages sent using postDelayed()
        }
    }
}

private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        if (msg.what == ADVANCE_MSG) {
            int i = 0;
            for (View key: mWidgetsToAdvance.keySet()) {
                final View v =
key.findViewById(mWidgetsToAdvance.get(key).autoAdvanceViewId);
                final int delay = mAdvanceStagger * i;
                if (v instanceof Advanceable) {
                    postDelayed(new Runnable() {
                        public void run() {
                            ((Advanceable) v).advance();
                        }
                    }, delay);
                }
                i++;
            }
            sendAdvanceMessage(mAdvanceInterval);
        }
    }
};

void addWidgetToAutoAdvanceIfNeeded(View hostView, AppWidgetProviderInfo
appWidgetInfo) {
    if (appWidgetInfo == null || appWidgetInfo.autoAdvanceViewId == -1) return;
    View v = hostView.findViewById(appWidgetInfo.autoAdvanceViewId);
    if (v instanceof Advanceable) {
        mWidgetsToAdvance.put(hostView, appWidgetInfo);
        ((Advanceable) v).fyiWillBeAdvancedByHostKThx();
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        updateRunning();
    }
}

void removeWidgetToAutoAdvance(View hostView) {
    if (mWidgetsToAdvance.containsKey(hostView)) {
        mWidgetsToAdvance.remove(hostView);
        updateRunning();
    }
}

public void removeAppWidget(LauncherAppWidgetInfo launcherInfo) {
    removeWidgetToAutoAdvance(launcherInfo.hostView);
    launcherInfo.hostView = null;
}

void showOutOfSpaceMessage(boolean isHotseatLayout) {
    int strId = (isHotseatLayout ? R.string.hotseat_out_of_space : R.string.out_of_space);
    Toast.makeText(this, getString(strId), Toast.LENGTH_SHORT).show();
}

public LauncherAppWidgetHost getAppWidgetHost() {
    return mAppWidgetHost;
}

public LauncherModel getModel() {
    return mModel;
}

void closeSystemDialogs() {
    getWindow().closeAllPanels();

    // Whatever we were doing is hereby canceled.
    mWaitingForResult = false;
}

@Override
protected void onNewIntent(Intent intent) {
    long startTime = 0;
    if (DEBUG_RESUME_TIME) {
        startTime = System.currentTimeMillis();
    }
    super.onNewIntent(intent);

    // Close the menu
    if (Intent.ACTION_MAIN.equals(intent.getAction())) {
        // also will cancel mWaitingForResult.
        closeSystemDialogs();

        final boolean alreadyOnHome =

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        ((intent.getFlags() & Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT)
         != Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);

Runnable processIntent = new Runnable() {
    public void run() {
        if (mWorkspace == null) {
            // Can be cases where mWorkspace is null, this prevents a NPE
            return;
        }
        Folder openFolder = mWorkspace.getOpenFolder();
        // In all these cases, only animate if we're already on home
        mWorkspace.exitWidgetResizeMode();
        if (alreadyOnHome && mState == State.WORKSPACE &&
!mWorkspace.isTouchActive() &&
            openFolder == null) {
            mWorkspace.moveToDefaultScreen(true);
        }

        closeFolder();
        exitSpringLoadedDragMode();

        // If we are already on home, then just animate back to the workspace,
        // otherwise, just wait until onResume to set the state back to Workspace
        if (alreadyOnHome) {
            showWorkspace(true);
        } else {
            mOnResumeState = State.WORKSPACE;
        }

        final View v = getWindow().peekDecorView();
        if (v != null && v.getWindowToken() != null) {
            InputMethodManager imm = (InputMethodManager) getSystemService(
                INPUT_METHOD_SERVICE);
            imm.hideSoftInputFromWindow(v.getWindowToken(), 0);
        }

        // Reset AllApps to its initial state
        if (!alreadyOnHome && mAppsCustomizeTabHost != null) {
            mAppsCustomizeTabHost.reset();
        }
    }
};

if (alreadyOnHome && !mWorkspace.hasWindowFocus()) {
    // Delay processing of the intent to allow the status bar animation to finish
    // first in order to avoid janky animations.
    mWorkspace.postDelayed(processIntent, 350);
} else {
    // Process the intent immediately.
    processIntent.run();

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    }
    if (DEBUG_RESUME_TIME) {
        Log.d(TAG, "Time spent in onNewIntent: " + (System.currentTimeMillis() - startTime));
    }
}

@Override
public void onRestoreInstanceState(Bundle state) {
    super.onRestoreInstanceState(state);
    for (int page: mSynchronouslyBoundPages) {
        mWorkspace.restoreInstanceStateForChild(page);
    }
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    outState.putInt(RUNTIME_STATE_CURRENT_SCREEN, mWorkspace.getNextPage());
    super.onSaveInstanceState(outState);

    outState.putInt(RUNTIME_STATE, mState.ordinal());
    // We close any open folder since it will not be re-opened, and we need to make sure
    // this state is reflected.
    closeFolder();

    if (mPendingAddInfo.container != ItemInfo.NO_ID && mPendingAddInfo.screen > -1 &&
        mWaitingForResult) {
        outState.putLong(RUNTIME_STATE_PENDING_ADD_CONTAINER,
mPendingAddInfo.container);
        outState.putInt(RUNTIME_STATE_PENDING_ADD_SCREEN,
mPendingAddInfo.screen);
        outState.putInt(RUNTIME_STATE_PENDING_ADD_CELL_X, mPendingAddInfo.cellX);
        outState.putInt(RUNTIME_STATE_PENDING_ADD_CELL_Y, mPendingAddInfo.cellY);
        outState.putInt(RUNTIME_STATE_PENDING_ADD_SPAN_X,
mPendingAddInfo.spanX);
        outState.putInt(RUNTIME_STATE_PENDING_ADD_SPAN_Y,
mPendingAddInfo.spanY);
        outState.putParcelable(RUNTIME_STATE_PENDING_ADD_WIDGET_INFO,
mPendingAddWidgetInfo);
        outState.putInt(RUNTIME_STATE_PENDING_ADD_WIDGET_ID,
mPendingAddWidgetId);
    }

    if (mFolderInfo != null && mWaitingForResult) {
        outState.putBoolean(RUNTIME_STATE_PENDING_FOLDER_RENAME, true);
        outState.putLong(RUNTIME_STATE_PENDING_FOLDER_RENAME_ID,
mFolderInfo.id);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

// Save the current AppsCustomize tab
if (mAppsCustomizeTabHost != null) {
    String currentTabTag = mAppsCustomizeTabHost.getCurrentTabTag();
    if (currentTabTag != null) {
        outState.putString("apps_customize_currentTab", currentTabTag);
    }
    int currentIndex = mAppsCustomizeContent.getSaveInstanceStateIndex();
    outState.putInt("apps_customize_currentIndex", currentIndex);
}
}

@Override
public void onDestroy() {
    super.onDestroy();

    // Remove all pending runnables
    mHandler.removeMessages(ADVANCE_MSG);
    mHandler.removeMessages(0);
    mWorkspace.removeCallbacks(mBuildLayersRunnable);

    // Stop callbacks from LauncherModel
    LauncherApplication app = ((LauncherApplication) getApplication());
    mModel.stopLoader();
    app.setLauncher(null);

    try {
        mAppWidgetHost.stopListening();
    } catch (NullPointerException ex) {
        Log.w(TAG, "problem while stopping AppWidgetHost during Launcher destruction", ex);
    }
    mAppWidgetHost = null;

    mWidgetsToAdvance.clear();

    TextKeyListener.getInstance().release();

    // Disconnect any of the callbacks and drawables associated with ItemInfos on the
workspace
    // to prevent leaking Launcher activities on orientation change.
    if (mModel != null) {
        mModel.unbindItemInfosAndClearQueuedBindRunnables();
    }

    getContentResolver().unregisterContentObserver(mWidgetObserver);
    unregisterReceiver(mCloseSystemDialogsReceiver);

    mDragLayer.clearAllResizeFrames();
    ((ViewGroup) mWorkspace.getParent()).removeAllViews();
    mWorkspace.removeAllViews();
    mWorkspace = null;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    mDragController = null;

    LauncherAnimUtils.onDestroyActivity();
}

public DragController getDragController() {
    return mDragController;
}

@Override
public void startActivityForResult(Intent intent, int requestCode) {
    if (requestCode >= 0) mWaitingForResult = true;
    super.startActivityForResult(intent, requestCode);
}

/**
 * Indicates that we want global search for this activity by setting the globalSearch
 * argument for {@link #startSearch} to true.
 */
@Override
public void startSearch(String initialQuery, boolean selectInitialQuery,
    Bundle appSearchData, boolean globalSearch) {

    showWorkspace(true);

    if (initialQuery == null) {
        // Use any text typed in the launcher as the initial query
        initialQuery = getTypedText();
    }
    if (appSearchData == null) {
        appSearchData = new Bundle();
        appSearchData.putString(Search.SOURCE, "launcher-search");
    }
    Rect sourceBounds = new Rect();
    if (mSearchDropTargetBar != null) {
        sourceBounds = mSearchDropTargetBar.getSearchBarBounds();
    }

    startGlobalSearch(initialQuery, selectInitialQuery,
        appSearchData, sourceBounds);
}

/**
 * Starts the global search activity. This code is a copied from SearchManager
 */
public void startGlobalSearch(String initialQuery,
    boolean selectInitialQuery, Bundle appSearchData, Rect sourceBounds) {
    final SearchManager searchManager =
        (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    ComponentName globalSearchActivity = searchManager.getGlobalSearchActivity();

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    if (globalSearchActivity == null) {
        Log.w(TAG, "No global search activity found.");
        return;
    }
    Intent intent = new Intent(SearchManager.INTENT_ACTION_GLOBAL_SEARCH);
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    intent.setComponent(globalSearchActivity);
    // Make sure that we have a Bundle to put source in
    if (appSearchData == null) {
        appSearchData = new Bundle();
    } else {
        appSearchData = new Bundle(appSearchData);
    }
    // Set source to package name of app that starts global search, if not set already.
    if (!appSearchData.containsKey("source")) {
        appSearchData.putString("source", getPackageName());
    }
    intent.putExtra(SearchManager.APP_DATA, appSearchData);
    if (!TextUtils.isEmpty(initialQuery)) {
        intent.putExtra(SearchManager.QUERY, initialQuery);
    }
    if (selectInitialQuery) {
        intent.putExtra(SearchManager.EXTRA_SELECT_QUERY, selectInitialQuery);
    }
    intent.setSourceBounds(sourceBounds);
    try {
        startActivity(intent);
    } catch (ActivityNotFoundException ex) {
        Log.e(TAG, "Global search activity not found: " + globalSearchActivity);
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    if (isWorkspaceLocked()) {
        return false;
    }

    super.onCreateOptionsMenu(menu);

    Intent manageApps = new
Intent(Settings.ACTION_MANAGE_ALL_APPLICATIONS_SETTINGS);
    manageApps.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
        | Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
    Intent settings = new Intent(android.provider.Settings.ACTION_SETTINGS);
    settings.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
        | Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED);
    String helpUrl = getString(R.string.help_url);
    Intent help = new Intent(Intent.ACTION_VIEW, Uri.parse(helpUrl));
    help.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        | Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);

        menu.add(MENU_GROUP_WALLPAPER, MENU_WALLPAPER_SETTINGS, 0,
R.string.menu_wallpaper)
            .setIcon(android.R.drawable.ic_menu_gallery)
            .setAlphabeticShortcut('W');
        menu.add(0, MENU_MANAGE_APPS, 0, R.string.menu_manage_apps)
            .setIcon(android.R.drawable.ic_menu_manage)
            .setIntent(manageApps)
            .setAlphabeticShortcut('M');
        menu.add(0, MENU_SYSTEM_SETTINGS, 0, R.string.menu_settings)
            .setIcon(android.R.drawable.ic_menu_preferences)
            .setIntent(settings)
            .setAlphabeticShortcut('P');
        if (!helpUrl.isEmpty()) {
            menu.add(0, MENU_HELP, 0, R.string.menu_help)
                .setIcon(android.R.drawable.ic_menu_help)
                .setIntent(help)
                .setAlphabeticShortcut('H');
        }
        return true;
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        super.onPrepareOptionsMenu(menu);

        if (mAppsCustomizeTabHost.isTransitioning()) {
            return false;
        }
        boolean allAppsVisible = (mAppsCustomizeTabHost.getVisibility() == View.VISIBLE);
        menu.setGroupVisible(MENU_GROUP_WALLPAPER, !allAppsVisible);

        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case MENU_WALLPAPER_SETTINGS:
                startWallpaper();
                return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @Override
    public boolean onSearchRequested() {
        startSearch(null, false, null, true);
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        // Use a custom animation for launching search
        return true;
    }

    public boolean isWorkspaceLocked() {
        return mWorkspaceLoading || mWaitingForResult;
    }

    private void resetAddInfo() {
        mPendingAddInfo.container = ItemInfo.NO_ID;
        mPendingAddInfo.screen = -1;
        mPendingAddInfo.cellX = mPendingAddInfo.cellY = -1;
        mPendingAddInfo.spanX = mPendingAddInfo.spanY = -1;
        mPendingAddInfo.minSpanX = mPendingAddInfo.minSpanY = -1;
        mPendingAddInfo.dropPos = null;
    }

    void addAppWidgetImpl(final int appWidgetId, ItemInfo info, AppWidgetHostView
boundWidget,
        AppWidgetProviderInfo appWidgetInfo) {
        if (appWidgetInfo.configure != null) {
            mPendingAddWidgetInfo = appWidgetInfo;
            mPendingAddWidgetId = appWidgetId;

            // Launch over to configure widget, if needed
            Intent intent = new Intent(AppWidgetManager.ACTION_APPWIDGET_CONFIGURE);
            intent.setComponent(appWidgetInfo.configure);
            intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
            startActivityForResultSafely(intent, REQUEST_CREATE_APPWIDGET);
        } else {
            // Otherwise just add it
            completeAddAppWidget(appWidgetId, info.container, info.screen, boundWidget,
                appWidgetInfo);
            // Exit spring loaded mode if necessary after adding the widget
            exitSpringLoadedDragModeDelayed(true, false, null);
        }
    }

    /**
     * Process a shortcut drop.
     *
     * @param componentName The name of the component
     * @param screen The screen where it should be added
     * @param cell The cell it should be added to, optional
     * @param position The location on the screen where it was dropped, optional
     */
    void processShortcutFromDrop(ComponentName componentName, long container, int
screen,
        int[] cell, int[] loc) {
        resetAddInfo();

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

mPendingAddInfo.container = container;
mPendingAddInfo.screen = screen;
mPendingAddInfo.dropPos = loc;

if (cell != null) {
    mPendingAddInfo.cellX = cell[0];
    mPendingAddInfo.cellY = cell[1];
}

Intent createShortcutIntent = new Intent(Intent.ACTION_CREATE_SHORTCUT);
createShortcutIntent.setComponent(componentName);
processShortcut(createShortcutIntent);
}

/**
 * Process a widget drop.
 *
 * @param info The PendingAppWidgetInfo of the widget being added.
 * @param screen The screen where it should be added
 * @param cell The cell it should be added to, optional
 * @param position The location on the screen where it was dropped, optional
 */
void addAppWidgetFromDrop(PendingAddWidgetInfo info, long container, int screen,
    int[] cell, int[] span, int[] loc) {
    resetAddInfo();
    mPendingAddInfo.container = info.container = container;
    mPendingAddInfo.screen = info.screen = screen;
    mPendingAddInfo.dropPos = loc;
    mPendingAddInfo.minSpanX = info.minSpanX;
    mPendingAddInfo.minSpanY = info.minSpanY;

    if (cell != null) {
        mPendingAddInfo.cellX = cell[0];
        mPendingAddInfo.cellY = cell[1];
    }
    if (span != null) {
        mPendingAddInfo.spanX = span[0];
        mPendingAddInfo.spanY = span[1];
    }

    AppWidgetHostView hostView = info.boundWidget;
    int appWidgetId;
    if (hostView != null) {
        appWidgetId = hostView.getAppWidgetId();
        addAppWidgetImpl(appWidgetId, info, hostView, info.info);
    } else {
        // In this case, we either need to start an activity to get permission to bind
        // the widget, or we need to start an activity to configure the widget, or both.
        appWidgetId = getAppWidgetHost().allocateAppWidgetId();
        Bundle options = info.bindOptions;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        boolean success = false;
        if (options != null) {
            success = mAppWidgetManager.bindAppWidgetIdIfAllowed(appWidgetId,
                info.componentName, options);
        } else {
            success = mAppWidgetManager.bindAppWidgetIdIfAllowed(appWidgetId,
                info.componentName);
        }
        if (success) {
            addAppWidgetImpl(appWidgetId, info, null, info.info);
        } else {
            mPendingAddWidgetInfo = info.info;
            Intent intent = new Intent(AppWidgetManager.ACTION_APPWIDGET_BIND);
            intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
            intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_PROVIDER,
info.componentName);
            // TODO: we need to make sure that this accounts for the options bundle.
            // intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_OPTIONS, options);
            startActivityResult(intent, REQUEST_BIND_APPWIDGET);
        }
    }
}

void processShortcut(Intent intent) {
    // Handle case where user selected "Applications"
    String applicationName = getResources().getString(R.string.group_applications);
    String shortcutName = intent.getStringExtra(Intent.EXTRA_SHORTCUT_NAME);

    if (applicationName != null && applicationName.equals(shortcutName)) {
        Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
        mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        Intent pickIntent = new Intent(Intent.ACTION_PICK_ACTIVITY);
        pickIntent.putExtra(Intent.EXTRA_INTENT, mainIntent);
        pickIntent.putExtra(Intent.EXTRA_TITLE, getText(R.string.title_select_application));
        startActivityResultSafely(pickIntent, REQUEST_PICK_APPLICATION);
    } else {
        startActivityResultSafely(intent, REQUEST_CREATE_SHORTCUT);
    }
}

void processWallpaper(Intent intent) {
    startActivityResult(intent, REQUEST_PICK_WALLPAPER);
}

FolderIcon addFolder(CellLayout layout, long container, final int screen, int cellX,
    int cellY) {
    final FolderInfo folderInfo = new FolderInfo();
    folderInfo.title = getText(R.string.folder_name);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        // Update the model
        LauncherModel.addToDatabase(Launcher.this, folderInfo, container, screen, cellX,
        cellY,
            false);
        sFolders.put(folderInfo.id, folderInfo);

        // Create the view
        FolderIcon newFolder =
            FolderIcon.fromXml(R.layout.folder_icon, this, layout, folderInfo, mIconCache);
        mWorkspace.addInScreen(newFolder, container, screen, cellX, cellY, 1, 1,
            isWorkspaceLocked());
        return newFolder;
    }

    void removeFolder(FolderInfo folder) {
        sFolders.remove(folder.id);
    }

    private void startWallpaper() {
        showWorkspace(true);
        final Intent pickWallpaper = new Intent(Intent.ACTION_SET_WALLPAPER);
        Intent chooser = Intent.createChooser(pickWallpaper,
            getText(R.string.chooser_wallpaper));
        // NOTE: Adds a configure option to the chooser if the wallpaper supports it
        //     Removed in Eclair MR1
        //     WallpaperManager wm = (WallpaperManager)
        //         getSystemService(Context.WALLPAPER_SERVICE);
        //     WallpaperInfo wi = wm.getWallpaperInfo();
        //     if (wi != null && wi.getSettingsActivity() != null) {
        //         LabeledIntent li = new LabeledIntent(getPackageName(),
        //             R.string.configure_wallpaper, 0);
        //         li.setClassName(wi.getPackageName(), wi.getSettingsActivity());
        //         chooser.putExtra(Intent.EXTRA_INITIAL_INTENTS, new Intent[] { li });
        //     }
        startActivityForResult(chooser, REQUEST_PICK_WALLPAPER);
    }

    /**
     * Registers various content observers. The current implementation registers
     * only a favorites observer to keep track of the favorites applications.
     */
    private void registerContentObservers() {
        ContentResolver resolver = getContentResolver();

        resolver.registerContentObserver(LauncherProvider.CONTENT_APPWIDGET_RESET_URI,
            true, mWidgetObserver);
    }

    @Override

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

public boolean dispatchKeyEvent(KeyEvent event) {
    if (event.getAction() == KeyEvent.ACTION_DOWN) {
        switch (event.getKeyCode()) {
            case KeyEvent.KEYCODE_HOME:
                return true;
            case KeyEvent.KEYCODE_VOLUME_DOWN:
                if (isPropertyEnabled(DUMP_STATE_PROPERTY)) {
                    dumpState();
                }
                return true;
            default:
                break;
        }
    } else if (event.getAction() == KeyEvent.ACTION_UP) {
        switch (event.getKeyCode()) {
            case KeyEvent.KEYCODE_HOME:
                return true;
            default:
                break;
        }
    }

    return super.dispatchKeyEvent(event);
}

@Override
public void onBackPressed() {
    if (isAllAppsVisible()) {
        showWorkspace(true);
    } else if (mWorkspace.getOpenFolder() != null) {
        Folder openFolder = mWorkspace.getOpenFolder();
        if (openFolder.isEditingName()) {
            openFolder.dismissEditingName();
        } else {
            closeFolder();
        }
    } else {
        mWorkspace.exitWidgetResizeMode();

        // Back button is a no-op here, but give at least some feedback for the button press
        mWorkspace.showOutlinesTemporarily();
    }
}

/**
 * Re-listen when widgets are reset.
 */
private void onAppWidgetReset() {
    if (mAppWidgetHost != null) {
        mAppWidgetHost.startListening();
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/**
 * Launches the intent referred by the clicked shortcut.
 *
 * @param v The view representing the clicked shortcut.
 */
public void onClick(View v) {
    // Make sure that rogue clicks don't get through while allapps is launching, or after the
    // view has detached (it's possible for this to happen if the view is removed mid touch).
    if (v.getWindowToken() == null) {
        return;
    }

    if (!mWorkspace.isFinishedSwitchingState()) {
        return;
    }

    Object tag = v.getTag();
    if (tag instanceof ShortcutInfo) {
        // Open shortcut
        final Intent intent = ((ShortcutInfo) tag).intent;
        int[] pos = new int[2];
        v.getLocationOnScreen(pos);
        intent.setSourceBounds(new Rect(pos[0], pos[1],
            pos[0] + v.getWidth(), pos[1] + v.getHeight()));

        boolean success = startActivitySafely(v, intent, tag);

        if (success && v instanceof BubbleTextView) {
            mWaitingForResume = (BubbleTextView) v;
            mWaitingForResume.setStayPressed(true);
        }
    } else if (tag instanceof FolderInfo) {
        if (v instanceof FolderIcon) {
            FolderIcon fi = (FolderIcon) v;
            handleFolderClick(fi);
        }
    } else if (v == mAllAppsButton) {
        if (isAllAppsVisible()) {
            showWorkspace(true);
        } else {
            onClickAllAppsButton(v);
        }
    }
}

public boolean onTouch(View v, MotionEvent event) {
    // this is an intercepted event being forwarded from mWorkspace;
    // clicking anywhere on the workspace causes the customization drawer to slide down
    showWorkspace(true);
    return false;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    /**
     * Event handler for the search button
     *
     * @param v The view that was clicked.
     */
    public void onClickSearchButton(View v) {
        v.performHapticFeedback(HapticFeedbackConstants.VIRTUAL_KEY);

        onSearchRequested();
    }

    /**
     * Event handler for the voice button
     *
     * @param v The view that was clicked.
     */
    public void onClickVoiceButton(View v) {
        v.performHapticFeedback(HapticFeedbackConstants.VIRTUAL_KEY);

        try {
            final SearchManager searchManager =
                (SearchManager) getSystemService(Context.SEARCH_SERVICE);
            ComponentName activityName = searchManager.getGlobalSearchActivity();
            Intent intent = new Intent(RecognizerIntent.ACTION_WEB_SEARCH);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            if (activityName != null) {
                intent.setPackage(activityName.getPackageName());
            }
            startActivity(null, intent, "onClickVoiceButton");
        } catch (ActivityNotFoundException e) {
            Intent intent = new Intent(RecognizerIntent.ACTION_WEB_SEARCH);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivitySafely(null, intent, "onClickVoiceButton");
        }
    }

    /**
     * Event handler for the "grid" button that appears on the home screen, which
     * enters all apps mode.
     *
     * @param v The view that was clicked.
     */
    public void onClickAllAppsButton(View v) {
        showAllApps(true);
    }

    public void onTouchDownAllAppsButton(View v) {
        // Provide the same haptic feedback that the system offers for virtual keys.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        v.performHapticFeedback(HapticFeedbackConstants.VIRTUAL_KEY);
    }

    public void onClickAppMarketButton(View v) {
        if (mAppMarketIntent != null) {
            startActivitySafely(v, mAppMarketIntent, "app market");
        } else {
            Log.e(TAG, "Invalid app market intent.");
        }
    }

    void startApplicationDetailsActivity(ComponentName componentName) {
        String packageName = componentName.getPackageName();
        Intent intent = new Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS,
            Uri.fromParts("package", packageName, null));
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
        startActivitySafely(null, intent, "startApplicationDetailsActivity");
    }

    void startApplicationUninstallActivity(ApplicationInfo appInfo) {
        if ((appInfo.flags & ApplicationInfo.DOWNLOADED_FLAG) == 0) {
            // System applications cannot be installed. For now, show a toast explaining that.
            // We may give them the option of disabling apps this way.
            int messageId = R.string.uninstall_system_app_text;
            Toast.makeText(this, messageId, Toast.LENGTH_SHORT).show();
        } else {
            String packageName = appInfo.componentName.getPackageName();
            String className = appInfo.componentName.getClassName();
            Intent intent = new Intent(
                Intent.ACTION_DELETE, Uri.fromParts("package", packageName, className));
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
                Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);
            startActivity(intent);
        }
    }

    boolean startActivity(View v, Intent intent, Object tag) {
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

        try {
            // Only launch using the new animation if the shortcut has not opted out (this is a
            // private contract between launcher and may be ignored in the future).
            boolean useLaunchAnimation = (v != null) &&
                !intent.hasExtra(INTENT_EXTRA_IGNORE_LAUNCH_ANIMATION);
            if (useLaunchAnimation) {
                ActivityOptions opts = ActivityOptions.makeScaleUpAnimation(v, 0, 0,
                    v.getMeasuredWidth(), v.getMeasuredHeight());

                startActivity(intent, opts.toBundle());
            }
        }
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        } else {
            startActivity(intent);
        }
        return true;
    } catch (SecurityException e) {
        Toast.makeText(this, R.string.activity_not_found, Toast.LENGTH_SHORT).show();
        Log.e(TAG, "Launcher does not have the permission to launch " + intent +
            ". Make sure to create a MAIN intent-filter for the corresponding activity " +
            "or use the exported attribute for this activity. "
            + "tag="+ tag + " intent=" + intent, e);
    }
    return false;
}

boolean startActivitySafely(View v, Intent intent, Object tag) {
    boolean success = false;
    try {
        success = startActivity(v, intent, tag);
    } catch (ActivityNotFoundException e) {
        Toast.makeText(this, R.string.activity_not_found, Toast.LENGTH_SHORT).show();
        Log.e(TAG, "Unable to launch. tag=" + tag + " intent=" + intent, e);
    }
    return success;
}

void startActivityForResultSafely(Intent intent, int requestCode) {
    try {
        startActivityForResult(intent, requestCode);
    } catch (ActivityNotFoundException e) {
        Toast.makeText(this, R.string.activity_not_found, Toast.LENGTH_SHORT).show();
    } catch (SecurityException e) {
        Toast.makeText(this, R.string.activity_not_found, Toast.LENGTH_SHORT).show();
        Log.e(TAG, "Launcher does not have the permission to launch " + intent +
            ". Make sure to create a MAIN intent-filter for the corresponding activity " +
            "or use the exported attribute for this activity.", e);
    }
}

private void handleFolderClick(FolderIcon folderIcon) {
    final FolderInfo info = folderIcon.getFolderInfo();
    Folder openFolder = mWorkspace.getFolderForTag(info);

    // If the folder info reports that the associated folder is open, then verify that
    // it is actually opened. There have been a few instances where this gets out of sync.
    if (info.opened && openFolder == null) {
        Log.d(TAG, "Folder info marked as open, but associated folder is not open. Screen: "
            + info.screen + " (" + info.cellX + ", " + info.cellY + ")");
        info.opened = false;
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    if (!info.opened && !folderIcon.getFolder().isDestroyed()) {
        // Close any open folder
        closeFolder();
        // Open the requested folder
        openFolder(folderIcon);
    } else {
        // Find the open folder...
        int folderScreen;
        if (openFolder != null) {
            folderScreen = mWorkspace.getPageForView(openFolder);
            // .. and close it
            closeFolder(openFolder);
            if (folderScreen != mWorkspace.getCurrentPage()) {
                // Close any folder open on the current screen
                closeFolder();
                // Pull the folder onto this screen
                openFolder(folderIcon);
            }
        }
    }
}

/**
 * This method draws the FolderIcon to an ImageView and then adds and positions that
 * ImageView
 * in the DragLayer in the exact absolute location of the original FolderIcon.
 */
private void copyFolderIconToImage(FolderIcon fi) {
    final int width = fi.getMeasuredWidth();
    final int height = fi.getMeasuredHeight();

    // Lazy load ImageView, Bitmap and Canvas
    if (mFolderIconImageView == null) {
        mFolderIconImageView = new ImageView(this);
    }
    if (mFolderIconBitmap == null || mFolderIconBitmap.getWidth() != width ||
        mFolderIconBitmap.getHeight() != height) {
        mFolderIconBitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
        mFolderIconCanvas = new Canvas(mFolderIconBitmap);
    }

    DragLayer.LayoutParams lp;
    if (mFolderIconImageView.getLayoutParams() instanceof DragLayer.LayoutParams) {
        lp = (DragLayer.LayoutParams) mFolderIconImageView.getLayoutParams();
    } else {
        lp = new DragLayer.LayoutParams(width, height);
    }

    // The layout from which the folder is being opened may be scaled, adjust the starting
    // view size by this scale factor.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        float scale = mDragLayer.getDescendantRectRelativeToSelf(fi,
mRectForFolderAnimation);
        lp.customPosition = true;
        lp.x = mRectForFolderAnimation.left;
        lp.y = mRectForFolderAnimation.top;
        lp.width = (int) (scale * width);
        lp.height = (int) (scale * height);

        mFolderIconCanvas.drawColor(0, PorterDuff.Mode.CLEAR);
        fi.draw(mFolderIconCanvas);
        mFolderIconImageView.setImageBitmap(mFolderIconBitmap);
        if (fi.getFolder() != null) {
            mFolderIconImageView.setPivotX(fi.getFolder().getPivotXForIconAnimation());
            mFolderIconImageView.setPivotY(fi.getFolder().getPivotYForIconAnimation());
        }
        // Just in case this image view is still in the drag layer from a previous animation,
        // we remove it and re-add it.
        if (mDragLayer.indexOfChild(mFolderIconImageView) != -1) {
            mDragLayer.removeView(mFolderIconImageView);
        }
        mDragLayer.addView(mFolderIconImageView, lp);
        if (fi.getFolder() != null) {
            fi.getFolder().bringToFront();
        }
    }

    private void growAndFadeOutFolderIcon(FolderIcon fi) {
        if (fi == null) return;
        PropertyValuesHolder alpha = PropertyValuesHolder.ofFloat("alpha", 0);
        PropertyValuesHolder scaleX = PropertyValuesHolder.ofFloat("scaleX", 1.5f);
        PropertyValuesHolder scaleY = PropertyValuesHolder.ofFloat("scaleY", 1.5f);

        FolderInfo info = (FolderInfo) fi.getTag();
        if (info.container == LauncherSettings.Favorites.CONTAINER_HOTSEAT) {
            CellLayout cl = (CellLayout) fi.getParent().getParent();
            CellLayout.LayoutParams lp = (CellLayout.LayoutParams) fi.getLayoutParams();
            cl.setFolderLeaveBehindCell(lp.cellX, lp.cellY);
        }

        // Push an ImageView copy of the FolderIcon into the DragLayer and hide the original
        copyFolderIconToImage(fi);
        fi.setVisibility(View.INVISIBLE);

        ObjectAnimator oa = LauncherAnimUtils.ofPropertyValuesHolder(mFolderIconImageView,
alpha,
        scaleX, scaleY);
        oa.setDuration(getResources().getInteger(R.integer.config_folderAnimDuration));
        oa.start();
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

private void shrinkAndFadeInFolderIcon(final FolderIcon fi) {
    if (fi == null) return;
    PropertyValuesHolder alpha = PropertyValuesHolder.ofFloat("alpha", 1.0f);
    PropertyValuesHolder scaleX = PropertyValuesHolder.ofFloat("scaleX", 1.0f);
    PropertyValuesHolder scaleY = PropertyValuesHolder.ofFloat("scaleY", 1.0f);

    final CellLayout cl = (CellLayout) fi.getParent().getParent();

    // We remove and re-draw the FolderIcon in-case it has changed
    mDragLayer.removeView(mFolderIconImageView);
    copyFolderIconToImage(fi);
    ObjectAnimator oa = LauncherAnimUtils.ofPropertyValuesHolder(mFolderIconImageView,
alpha,
    scaleX, scaleY);
    oa.setDuration(getResources().getInteger(R.integer.config_folderAnimDuration));
    oa.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationEnd(Animator animation) {
            if (cl != null) {
                cl.clearFolderLeaveBehind();
                // Remove the ImageView copy of the FolderIcon and make the original visible.
                mDragLayer.removeView(mFolderIconImageView);
                fi.setVisibility(View.VISIBLE);
            }
        }
    });
    oa.start();
}

/**
 * Opens the user folder described by the specified tag. The opening of the folder
 * is animated relative to the specified View. If the View is null, no animation
 * is played.
 *
 * @param folderInfo The FolderInfo describing the folder to open.
 */
public void openFolder(FolderIcon folderIcon) {
    Folder folder = folderIcon.getFolder();
    FolderInfo info = folder.mInfo;

    info.opened = true;

    // Just verify that the folder hasn't already been added to the DragLayer.
    // There was a one-off crash where the folder had a parent already.
    if (folder.getParent() == null) {
        mDragLayer.addView(folder);
        mDragController.addDropTarget((DropTarget) folder);
    } else {
        Log.w(TAG, "Opening folder (" + folder + ") which already has a parent (" +
            folder.getParent() + ").");
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }
    folder.animateOpen();
    growAndFadeOutFolderIcon(folderIcon);

    // Notify the accessibility manager that this folder "window" has appeared and occluded
    // the workspace items
    folder.sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);

getDragLayer().sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_CONTENT_CHANGED);
}

public void closeFolder() {
    Folder folder = mWorkspace.getOpenFolder();
    if (folder != null) {
        if (folder.isEditingName()) {
            folder.dismissEditingName();
        }
        closeFolder(folder);

        // Dismiss the folder cling
        dismissFolderCling(null);
    }
}

void closeFolder(Folder folder) {
    folder.getInfo().opened = false;

    ViewGroup parent = (ViewGroup) folder.getParent().getParent();
    if (parent != null) {
        FolderIcon fi = (FolderIcon) mWorkspace.getViewForTag(folder.mInfo);
        shrinkAndFadeInFolderIcon(fi);
    }
    folder.animateClosed();

    // Notify the accessibility manager that this folder "window" has disappeared and no
    // longer occludes the workspace items

getDragLayer().sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);
}

public boolean onLongClick(View v) {
    if (!isDraggingEnabled()) return false;
    if (isWorkspaceLocked()) return false;
    if (mState != State.WORKSPACE) return false;

    if (!(v instanceof CellLayout)) {
        v = (View) v.getParent().getParent();
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        resetAddInfo();
        CellLayout.CellInfo longClickCellInfo = (CellLayout.CellInfo) v.getTag();
        // This happens when long clicking an item with the dpad/trackball
        if (longClickCellInfo == null) {
            return true;
        }

        // The hotseat touch handling does not go through Workspace, and we always allow long
press
        // on hotseat items.
        final View itemUnderLongClick = longClickCellInfo.cell;
        boolean allowLongPress = isHotseatLayout(v) || mWorkspace.allowLongPress();
        if (allowLongPress && !mDragController.isDragging()) {
            if (itemUnderLongClick == null) {
                // User long pressed on empty space
                mWorkspace.performHapticFeedback(HapticFeedbackConstants.LONG_PRESS,
                    HapticFeedbackConstants.FLAG_IGNORE_VIEW_SETTING);
                startWallpaper();
            } else {
                if (!(itemUnderLongClick instanceof Folder)) {
                    // User long pressed on an item
                    mWorkspace.startDrag(longClickCellInfo);
                }
            }
        }
        return true;
    }

    boolean isHotseatLayout(View layout) {
        return mHotseat != null && layout != null &&
            (layout instanceof CellLayout) && (layout == mHotseat.getLayout());
    }

    Hotseat getHotseat() {
        return mHotseat;
    }

    SearchDropTargetBar getSearchBar() {
        return mSearchDropTargetBar;
    }

    /**
     * Returns the CellLayout of the specified container at the specified screen.
     */
    CellLayout getCellLayout(long container, int screen) {
        if (container == LauncherSettings.Favorites.CONTAINER_HOTSEAT) {
            if (mHotseat != null) {
                return mHotseat.getLayout();
            } else {
                return null;
            }
        }
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    } else {
        return (CellLayout) mWorkspace.getChildAt(screen);
    }
}

Workspace getWorkspace() {
    return mWorkspace;
}

// Now a part of LauncherModel.Callbacks. Used to reorder loading steps.
@Override
public boolean isAllAppsVisible() {
    return (mState == State.APPS_CUSTOMIZE) || (mOnResumeState ==
State.APPS_CUSTOMIZE);
}

@Override
public boolean isAllAppsButtonRank(int rank) {
    return mHotseat.isAllAppsButtonRank(rank);
}

/**
 * Helper method for the cameraZoomIn/cameraZoomOut animations
 * @param view The view being animated
 * @param scaleFactor The scale factor used for the zoom
 */
private void setPivotsForZoom(View view, float scaleFactor) {
    view.setPivotX(view.getWidth() / 2.0f);
    view.setPivotY(view.getHeight() / 2.0f);
}

void disableWallpaperIfInAllApps() {
    // Only disable it if we are in all apps
    if (isAllAppsVisible()) {
        if (mAppsCustomizeTabHost != null &&
            !mAppsCustomizeTabHost.isTransitioning()) {
            updateWallpaperVisibility(false);
        }
    }
}

private void setWorkspaceBackground(boolean workspace) {
    mLauncherView.setBackground(workspace ?
        mWorkspaceBackgroundDrawable : null);
}

void updateWallpaperVisibility(boolean visible) {
    int wpflags = visible ? WindowManager.LayoutParams.FLAG_SHOW_WALLPAPER : 0;
    int curflags = getWindow().getAttributes().flags
& WindowManager.LayoutParams.FLAG_SHOW_WALLPAPER;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        if (wpflags != curflags) {
            getWindow().setFlags(wpflags,
WindowManager.LayoutParams.FLAG_SHOW_WALLPAPER);
        }
        setWorkspaceBackground(visible);
    }

    private void dispatchOnLauncherTransitionPrepare(View v, boolean animated, boolean
toWorkspace) {
        if (v instanceof LauncherTransitionable) {
            ((LauncherTransitionable) v).onLauncherTransitionPrepare(this, animated,
toWorkspace);
        }
    }

    private void dispatchOnLauncherTransitionStart(View v, boolean animated, boolean
toWorkspace) {
        if (v instanceof LauncherTransitionable) {
            ((LauncherTransitionable) v).onLauncherTransitionStart(this, animated, toWorkspace);
        }

        // Update the workspace transition step as well
        dispatchOnLauncherTransitionStep(v, 0f);
    }

    private void dispatchOnLauncherTransitionStep(View v, float t) {
        if (v instanceof LauncherTransitionable) {
            ((LauncherTransitionable) v).onLauncherTransitionStep(this, t);
        }
    }

    private void dispatchOnLauncherTransitionEnd(View v, boolean animated, boolean
toWorkspace) {
        if (v instanceof LauncherTransitionable) {
            ((LauncherTransitionable) v).onLauncherTransitionEnd(this, animated, toWorkspace);
        }

        // Update the workspace transition step as well
        dispatchOnLauncherTransitionStep(v, 1f);
    }

/**
 * Things to test when changing the following seven functions.
 * - Home from workspace
 *     - from center screen
 *     - from other screens
 * - Home from all apps
 *     - from center screen
 *     - from other screens
 * - Back from all apps

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

*      - from center screen
*      - from other screens
* - Launch app from workspace and quit
*      - with back
*      - with home
* - Launch app from all apps and quit
*      - with back
*      - with home
* - Go to a screen that's not the default, then all
*   apps, and launch an app, and go back
*      - with back
*      -with home
* - On workspace, long press power and go back
*      - with back
*      - with home
* - On all apps, long press power and go back
*      - with back
*      - with home
* - On workspace, power off
* - On all apps, power off
* - Launch an app and turn off the screen while in that app
*      - Go back with home key
*      - Go back with back key TODO: make this not go to workspace
*      - From all apps
*      - From workspace
* - Enter and exit car mode (because it causes an extra configuration changed)
*      - From all apps
*      - From the center workspace
*      - From another workspace
*/

```

```

/**

```

```

* Zoom the camera out from the workspace to reveal 'toView'.
* Assumes that the view to show is anchored at either the very top or very bottom
* of the screen.
*/

```

```

private void showAppsCustomizeHelper(final boolean animated, final boolean springLoaded)
{
    if (mStateAnimation != null) {
        mStateAnimation.setDuration(0);
        mStateAnimation.cancel();
        mStateAnimation = null;
    }
    final Resources res = getResources();

    final int duration = res.getInteger(R.integer.config_appsCustomizeZoomInTime);
    final int fadeDuration = res.getInteger(R.integer.config_appsCustomizeFadeInTime);
    final float scale = (float) res.getInteger(R.integer.config_appsCustomizeZoomScaleFactor);
    final View fromView = mWorkspace;
    final AppsCustomizeTabHost toView = mAppsCustomizeTabHost;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

final int startDelay =
    res.getInteger(R.integer.config_workspaceAppsCustomizeAnimationStagger);

setPivotsForZoom(toView, scale);

// Shrink workspaces away if going to AppsCustomize from workspace
Animator workspaceAnim =
    mWorkspace.getStateAnimation(Workspace.State.SMALL, animated);

if (animated) {
    toView.setScaleX(scale);
    toView.setScaleY(scale);
    final LauncherViewPropertyAnimator scaleAnim = new
LauncherViewPropertyAnimator(toView);
    scaleAnim.
        scaleX(1f).scaleY(1f).
        setDuration(duration).
        setInterpolator(new Workspace.ZoomOutInterpolator());

    toView.setVisibility(View.VISIBLE);
    toView.setAlpha(0f);
    final ObjectAnimator alphaAnim = LauncherAnimUtils
        .ofFloat(toView, "alpha", 0f, 1f)
        .setDuration(fadeDuration);
    alphaAnim.setInterpolator(new DecelerateInterpolator(1.5f));
    alphaAnim.addListener(new AnimatorUpdateListener() {
        @Override
        public void onAnimationUpdate(ValueAnimator animation) {
            if (animation == null) {
                throw new RuntimeException("animation is null");
            }
            float t = (Float) animation.getAnimatedValue();
            dispatchOnLauncherTransitionStep(fromView, t);
            dispatchOnLauncherTransitionStep(toView, t);
        }
    });

    // toView should appear right at the end of the workspace shrink
    // animation
    mStateAnimation = LauncherAnimUtils.createAnimatorSet();
    mStateAnimation.play(scaleAnim).after(startDelay);
    mStateAnimation.play(alphaAnim).after(startDelay);

    mStateAnimation.addListener(new AnimatorListenerAdapter() {
        boolean animationCancelled = false;

        @Override
        public void onAnimationStart(Animator animation) {
            updateWallpaperVisibility(true);
            // Prepare the position

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        toView.setTranslationX(0.0f);
        toView.setTranslationY(0.0f);
        toView.setVisibility(View.VISIBLE);
        toView.bringToFront();
    }
    @Override
    public void onAnimationEnd(Animator animation) {
        dispatchOnLauncherTransitionEnd(fromView, animated, false);
        dispatchOnLauncherTransitionEnd(toView, animated, false);

        if (mWorkspace != null && !springLoaded &&
!LauncherApplication.isScreenLarge()) {
            // Hide the workspace scrollbar
            mWorkspace.hideScrollingIndicator(true);
            hideDockDivider();
        }
        if (!animationCancelled) {
            updateWallpaperVisibility(false);
        }

        // Hide the search bar
        if (mSearchDropTargetBar != null) {
            mSearchDropTargetBar.hideSearchBar(false);
        }
    }

    @Override
    public void onAnimationCancel(Animator animation) {
        animationCancelled = true;
    }
});

if (workspaceAnim != null) {
    mStateAnimation.play(workspaceAnim);
}

boolean delayAnim = false;

dispatchOnLauncherTransitionPrepare(fromView, animated, false);
dispatchOnLauncherTransitionPrepare(toView, animated, false);

// If any of the objects being animated haven't been measured/laid out
// yet, delay the animation until we get a layout pass
if (((LauncherTransitionable) toView).getContent().getMeasuredWidth() == 0) ||
    (mWorkspace.getMeasuredWidth() == 0) ||
    (toView.getMeasuredWidth() == 0)) {
    delayAnim = true;
}

final AnimatorSet stateAnimation = mStateAnimation;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

final Runnable startAnimRunnable = new Runnable() {
    public void run() {
        // Check that mStateAnimation hasn't changed while
        // we waited for a layout/draw pass
        if (mStateAnimation != stateAnimation)
            return;
        setPivotsForZoom(toView, scale);
        dispatchOnLauncherTransitionStart(fromView, animated, false);
        dispatchOnLauncherTransitionStart(toView, animated, false);
        LauncherAnimUtils.startAnimationAfterNextDraw(mStateAnimation, toView);
    }
};
if (delayAnim) {
    final ViewTreeObserver observer = toView.getViewTreeObserver();
    observer.addOnGlobalLayoutListener(new OnGlobalLayoutListener() {
        public void onGlobalLayout() {
            startAnimRunnable.run();
            toView.getViewTreeObserver().removeOnGlobalLayoutListener(this);
        }
    });
} else {
    startAnimRunnable.run();
}
} else {
    toView.setTranslationX(0.0f);
    toView.setTranslationY(0.0f);
    toView.setScaleX(1.0f);
    toView.setScaleY(1.0f);
    toView.setVisibility(View.VISIBLE);
    toView.bringToFront();

    if (!springLoaded && !LauncherApplication.isScreenLarge()) {
        // Hide the workspace scrollbar
        mWorkspace.hideScrollingIndicator(true);
        hideDockDivider();

        // Hide the search bar
        if (mSearchDropTargetBar != null) {
            mSearchDropTargetBar.hideSearchBar(false);
        }
    }
    dispatchOnLauncherTransitionPrepare(fromView, animated, false);
    dispatchOnLauncherTransitionStart(fromView, animated, false);
    dispatchOnLauncherTransitionEnd(fromView, animated, false);
    dispatchOnLauncherTransitionPrepare(toView, animated, false);
    dispatchOnLauncherTransitionStart(toView, animated, false);
    dispatchOnLauncherTransitionEnd(toView, animated, false);
    updateWallpaperVisibility(false);
}
}
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/**
 * Zoom the camera back into the workspace, hiding 'fromView'.
 * This is the opposite of showAppsCustomizeHelper.
 * @param animated If true, the transition will be animated.
 */
private void hideAppsCustomizeHelper(State toState, final boolean animated,
    final boolean springLoaded, final Runnable onCompleteRunnable) {

    if (mStateAnimation != null) {
        mStateAnimation.setDuration(0);
        mStateAnimation.cancel();
        mStateAnimation = null;
    }
    Resources res = getResources();

    final int duration = res.getInteger(R.integer.config_appsCustomizeZoomOutTime);
    final int fadeOutDuration =
        res.getInteger(R.integer.config_appsCustomizeFadeOutTime);
    final float scaleFactor = (float)
        res.getInteger(R.integer.config_appsCustomizeZoomScaleFactor);
    final View fromView = mAppsCustomizeTabHost;
    final View toView = mWorkspace;
    Animator workspaceAnim = null;

    if (toState == State.WORKSPACE) {
        int stagger =
res.getInteger(R.integer.config_appsCustomizeWorkspaceAnimationStagger);
        workspaceAnim = mWorkspace.getChangeStateAnimation(
            Workspace.State.NORMAL, animated, stagger);
    } else if (toState == State.APPS_CUSTOMIZE_SPRING_LOADED) {
        workspaceAnim = mWorkspace.getChangeStateAnimation(
            Workspace.State.SPRING_LOADED, animated);
    }

    setPivotsForZoom(fromView, scaleFactor);
    updateWallpaperVisibility(true);
    showHotseat(animated);
    if (animated) {
        final LauncherViewPropertyAnimator scaleAnim =
            new LauncherViewPropertyAnimator(fromView);
        scaleAnim.
            scaleX(scaleFactor).scaleY(scaleFactor).
            setDuration(duration).
            setInterpolator(new Workspace.ZoomInInterpolator());

        final ObjectAnimator alphaAnim = LauncherAnimUtils
            .ofFloat(fromView, "alpha", 1f, 0f)
            .setDuration(fadeOutDuration);
        alphaAnim.setInterpolator(new AccelerateDecelerateInterpolator());
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

alphaAnim.addUpdateListener(new AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator animation) {
        float t = 1f - (Float) animation.getAnimatedValue();
        dispatchOnLauncherTransitionStep(fromView, t);
        dispatchOnLauncherTransitionStep(toView, t);
    }
});

mStateAnimation = LauncherAnimUtils.createAnimatorSet();

dispatchOnLauncherTransitionPrepare(fromView, animated, true);
dispatchOnLauncherTransitionPrepare(toView, animated, true);
mAppsCustomizeContent.pauseScrolling();

mStateAnimation.addListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        updateWallpaperVisibility(true);
        fromView.setVisibility(View.GONE);
        dispatchOnLauncherTransitionEnd(fromView, animated, true);
        dispatchOnLauncherTransitionEnd(toView, animated, true);
        if (mWorkspace != null) {
            mWorkspace.hideScrollingIndicator(false);
        }
        if (onCompleteRunnable != null) {
            onCompleteRunnable.run();
        }
        mAppsCustomizeContent.updateCurrentPageScroll();
        mAppsCustomizeContent.resumeScrolling();
    }
});

mStateAnimation.playTogether(scaleAnim, alphaAnim);
if (workspaceAnim != null) {
    mStateAnimation.play(workspaceAnim);
}
dispatchOnLauncherTransitionStart(fromView, animated, true);
dispatchOnLauncherTransitionStart(toView, animated, true);
LauncherAnimUtils.startAnimationAfterNextDraw(mStateAnimation, toView);
} else {
    fromView.setVisibility(View.GONE);
    dispatchOnLauncherTransitionPrepare(fromView, animated, true);
    dispatchOnLauncherTransitionStart(fromView, animated, true);
    dispatchOnLauncherTransitionEnd(fromView, animated, true);
    dispatchOnLauncherTransitionPrepare(toView, animated, true);
    dispatchOnLauncherTransitionStart(toView, animated, true);
    dispatchOnLauncherTransitionEnd(toView, animated, true);
    mWorkspace.hideScrollingIndicator(false);
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    @Override
    public void onTrimMemory(int level) {
        super.onTrimMemory(level);
        if (level >= ComponentCallbacks2.TRIM_MEMORY_MODERATE) {
            mAppsCustomizeTabHost.onTrimMemory();
        }
    }

    @Override
    public void onWindowFocusChanged(boolean hasFocus) {
        if (!hasFocus) {
            // When another window occludes launcher (like the notification shade, or recents),
            // ensure that we enable the wallpaper flag so that transitions are done correctly.
            updateWallpaperVisibility(true);
        } else {
            // When launcher has focus again, disable the wallpaper if we are in AllApps
            mWorkspace.postDelayed(new Runnable() {
                @Override
                public void run() {
                    disableWallpaperIfInAllApps();
                }
            }, 500);
        }
    }

    void showWorkspace(boolean animated) {
        showWorkspace(animated, null);
    }

    void showWorkspace(boolean animated, Runnable onCompleteRunnable) {
        if (mState != State.WORKSPACE) {
            boolean wasInSpringLoadedMode = (mState ==
State.APPS_CUSTOMIZE_SPRING_LOADED);
            mWorkspace.setVisibility(View.VISIBLE);
            hideAppsCustomizeHelper(State.WORKSPACE, animated, false,
onCompleteRunnable);

            // Show the search bar (only animate if we were showing the drop target bar in spring
            // loaded mode)
            if (mSearchDropTargetBar != null) {
                mSearchDropTargetBar.showSearchBar(wasInSpringLoadedMode);
            }

            // We only need to animate in the dock divider if we're going from spring loaded mode
            showDockDivider(animated && wasInSpringLoadedMode);

            // Set focus to the AppsCustomize button
            if (mAllAppsButton != null) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        mAllAppsButton.requestFocus();
    }
}

mWorkspace.flashScrollingIndicator(animated);

// Change the state *after* we've called all the transition code
mState = State.WORKSPACE;

// Resume the auto-advance of widgets
mUserPresent = true;
updateRunning();

// Send an accessibility event to announce the context change
getWindow().getDecorView()
    .sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);
}

void showAllApps(boolean animated) {
    if (mState != State.WORKSPACE) return;

    showAppsCustomizeHelper(animated, false);
    mAppsCustomizeTabHost.requestFocus();

    // Change the state *after* we've called all the transition code
    mState = State.APPS_CUSTOMIZE;

    // Pause the auto-advance of widgets until we are out of AllApps
    mUserPresent = false;
    updateRunning();
    closeFolder();

    // Send an accessibility event to announce the context change
    getWindow().getDecorView()
        .sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);
}

void enterSpringLoadedDragMode() {
    if (isAllAppsVisible()) {
        hideAppsCustomizeHelper(State.APPS_CUSTOMIZE_SPRING_LOADED, true, true,
null);
        hideDockDivider();
        mState = State.APPS_CUSTOMIZE_SPRING_LOADED;
    }
}

void exitSpringLoadedDragModeDelayed(final boolean successfulDrop, boolean
extendedDelay,
    final Runnable onCompleteRunnable) {
    if (mState != State.APPS_CUSTOMIZE_SPRING_LOADED) return;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

mHandler.postDelayed(new Runnable() {
    @Override
    public void run() {
        if (successfulDrop) {
            // Before we show workspace, hide all apps again because
            // exitSpringLoadedDragMode made it visible. This is a bit hacky; we should
            // clean up our state transition functions
            mAppsCustomizeTabHost.setVisibility(View.GONE);
            showWorkspace(true, onCompleteRunnable);
        } else {
            exitSpringLoadedDragMode();
        }
    }
}, (extendedDelay ?
    EXIT_SPRINGLOADED_MODE_LONG_TIMEOUT :
    EXIT_SPRINGLOADED_MODE_SHORT_TIMEOUT));
}

void exitSpringLoadedDragMode() {
    if (mState == State.APPS_CUSTOMIZE_SPRING_LOADED) {
        final boolean animated = true;
        final boolean springLoaded = true;
        showAppsCustomizeHelper(animated, springLoaded);
        mState = State.APPS_CUSTOMIZE;
    }
    // Otherwise, we are not in spring loaded mode, so don't do anything.
}

void hideDockDivider() {
    if (mQsbDivider != null && mDockDivider != null) {
        mQsbDivider.setVisibility(View.INVISIBLE);
        mDockDivider.setVisibility(View.INVISIBLE);
    }
}

void showDockDivider(boolean animated) {
    if (mQsbDivider != null && mDockDivider != null) {
        mQsbDivider.setVisibility(View.VISIBLE);
        mDockDivider.setVisibility(View.VISIBLE);
        if (mDividerAnimator != null) {
            mDividerAnimator.cancel();
            mQsbDivider.setAlpha(1f);
            mDockDivider.setAlpha(1f);
            mDividerAnimator = null;
        }
        if (animated) {
            mDividerAnimator = LauncherAnimUtils.createAnimatorSet();
            mDividerAnimator.playTogether(LauncherAnimUtils.ofFloat(mQsbDivider, "alpha", 1f),
                LauncherAnimUtils.ofFloat(mDockDivider, "alpha", 1f));
        }
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        int duration = 0;
        if (mSearchDropTargetBar != null) {
            duration = mSearchDropTargetBar.getTransitionInDuration();
        }
        mDividerAnimator.setDuration(duration);
        mDividerAnimator.start();
    }
}

void lockAllApps() {
    // TODO
}

void unlockAllApps() {
    // TODO
}

/**
 * Shows the hotseat area.
 */
void showHotseat(boolean animated) {
    if (!LauncherApplication.isScreenLarge()) {
        if (animated) {
            if (mHotseat.getAlpha() != 1f) {
                int duration = 0;
                if (mSearchDropTargetBar != null) {
                    duration = mSearchDropTargetBar.getTransitionInDuration();
                }
                mHotseat.animate().alpha(1f).setDuration(duration);
            }
        } else {
            mHotseat.setAlpha(1f);
        }
    }
}

/**
 * Hides the hotseat area.
 */
void hideHotseat(boolean animated) {
    if (!LauncherApplication.isScreenLarge()) {
        if (animated) {
            if (mHotseat.getAlpha() != 0f) {
                int duration = 0;
                if (mSearchDropTargetBar != null) {
                    duration = mSearchDropTargetBar.getTransitionOutDuration();
                }
                mHotseat.animate().alpha(0f).setDuration(duration);
            }
        }
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        } else {
            mHotseat.setAlpha(0f);
        }
    }
}

/**
 * Add an item from all apps or customize onto the given workspace screen.
 * If layout is null, add to the current screen.
 */
void addExternalItemToScreen(ItemInfo itemInfo, final CellLayout layout) {
    if (!mWorkspace.addExternalItemToScreen(itemInfo, layout)) {
        showOutOfSpaceMessage(isHotseatLayout(layout));
    }
}

/** Maps the current orientation to an index for referencing orientation correct global icons */
private int getCurrentOrientationIndexForGlobalIcons() {
    // default - 0, landscape - 1
    switch (getResources().getConfiguration().orientation) {
        case Configuration.ORIENTATION_LANDSCAPE:
            return 1;
        default:
            return 0;
    }
}

private Drawable getExternalPackageToolbarIcon(ComponentName activityName, String
resourceName) {
    try {
        PackageManager packageManager = getPackageManager();
        // Look for the toolbar icon specified in the activity meta-data
        Bundle metaData = packageManager.getActivityInfo(
            activityName, PackageManager.GET_META_DATA).metaData;
        if (metaData != null) {
            int iconResId = metaData.getInt(resourceName);
            if (iconResId != 0) {
                Resources res = packageManager.getResourcesForActivity(activityName);
                return res.getDrawable(iconResId);
            }
        }
    } catch (NameNotFoundException e) {
        // This can happen if the activity defines an invalid drawable
        Log.w(TAG, "Failed to load toolbar icon; " + activityName.flattenToShortString() +
            " not found", e);
    } catch (Resources.NotFoundException nfe) {
        // This can happen if the activity defines an invalid drawable
        Log.w(TAG, "Failed to load toolbar icon from " + activityName.flattenToShortString(),
            nfe);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    return null;
}

// if successful in getting icon, return it; otherwise, set button to use default drawable
private Drawable.ConstantState updateTextButtonWithIconFromExternalActivity(
    int buttonId, ComponentName activityName, int fallbackDrawableId,
    String toolbarResourceName) {
    Drawable toolbarIcon = getExternalPackageToolbarIcon(activityName,
toolbarResourceName);
    Resources r = getResources();
    int w = r.getDimensionPixelSize(R.dimen.toolbar_external_icon_width);
    int h = r.getDimensionPixelSize(R.dimen.toolbar_external_icon_height);

    TextView button = (TextView) findViewById(buttonId);
    // If we were unable to find the icon via the meta-data, use a generic one
    if (toolbarIcon == null) {
        toolbarIcon = r.getDrawable(fallbackDrawableId);
        toolbarIcon.setBounds(0, 0, w, h);
        if (button != null) {
            button.setCompoundDrawables(toolbarIcon, null, null, null);
        }
        return null;
    } else {
        toolbarIcon.setBounds(0, 0, w, h);
        if (button != null) {
            button.setCompoundDrawables(toolbarIcon, null, null, null);
        }
        return toolbarIcon.getConstantState();
    }
}

// if successful in getting icon, return it; otherwise, set button to use default drawable
private Drawable.ConstantState updateButtonWithIconFromExternalActivity(
    int buttonId, ComponentName activityName, int fallbackDrawableId,
    String toolbarResourceName) {
    ImageView button = (ImageView) findViewById(buttonId);
    Drawable toolbarIcon = getExternalPackageToolbarIcon(activityName,
toolbarResourceName);

    if (button != null) {
        // If we were unable to find the icon via the meta-data, use a
        // generic one
        if (toolbarIcon == null) {
            button.setImageResource(fallbackDrawableId);
        } else {
            button.setImageDrawable(toolbarIcon);
        }
    }

    return toolbarIcon != null ? toolbarIcon.getConstantState() : null;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    private void updateTextButtonWithDrawable(int buttonId, Drawable d) {
        TextView button = (TextView) findViewById(buttonId);
        button.setCompoundDrawables(d, null, null, null);
    }

    private void updateButtonWithDrawable(int buttonId, Drawable.ConstantState d) {
        ImageView button = (ImageView) findViewById(buttonId);
        button.setImageDrawable(d.newDrawable(getResources()));
    }

    private void invalidatePressedFocusedStates(View container, View button) {
        if (container instanceof HolographicLinearLayout) {
            HolographicLinearLayout layout = (HolographicLinearLayout) container;
            layout.invalidatePressedFocusedStates();
        } else if (button instanceof HolographicImageView) {
            HolographicImageView view = (HolographicImageView) button;
            view.invalidatePressedFocusedStates();
        }
    }

    private boolean updateGlobalSearchIcon() {
        final View searchButtonContainer = findViewById(R.id.search_button_container);
        final ImageView searchButton = (ImageView) findViewById(R.id.search_button);
        final View voiceButtonContainer = findViewById(R.id.voice_button_container);
        final View voiceButton = findViewById(R.id.voice_button);
        final View voiceButtonProxy = findViewById(R.id.voice_button_proxy);

        final SearchManager searchManager =
            (SearchManager) getSystemService(Context.SEARCH_SERVICE);
        ComponentName activityName = searchManager.getGlobalSearchActivity();
        if (activityName != null) {
            int coi = getCurrentOrientationIndexForGlobalIcons();
            sGlobalSearchIcon[coi] = updateButtonWithIconFromExternalActivity(
                R.id.search_button, activityName, R.drawable.ic_home_search_normal_holo,
                TOOLBAR_SEARCH_ICON_METADATA_NAME);
            if (sGlobalSearchIcon[coi] == null) {
                sGlobalSearchIcon[coi] = updateButtonWithIconFromExternalActivity(
                    R.id.search_button, activityName, R.drawable.ic_home_search_normal_holo,
                    TOOLBAR_ICON_METADATA_NAME);
            }

            if (searchButtonContainer != null) searchButtonContainer.setVisibility(View.VISIBLE);
            searchButton.setVisibility(View.VISIBLE);
            invalidatePressedFocusedStates(searchButtonContainer, searchButton);
            return true;
        } else {
            // We disable both search and voice search when there is no global search provider

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        if (searchButtonContainer != null) searchButtonContainer.setVisibility(View.GONE);
        if (voiceButtonContainer != null) voiceButtonContainer.setVisibility(View.GONE);
        searchButton.setVisibility(View.GONE);
        voiceButton.setVisibility(View.GONE);
        if (voiceButtonProxy != null) {
            voiceButtonProxy.setVisibility(View.GONE);
        }
        return false;
    }
}

private void updateGlobalSearchIcon(Drawable.ConstantState d) {
    final View searchButtonContainer = findViewById(R.id.search_button_container);
    final View searchButton = (ImageView) findViewById(R.id.search_button);
    updateButtonWithDrawable(R.id.search_button, d);
    invalidatePressedFocusedStates(searchButtonContainer, searchButton);
}

private boolean updateVoiceSearchIcon(boolean searchVisible) {
    final View voiceButtonContainer = findViewById(R.id.voice_button_container);
    final View voiceButton = findViewById(R.id.voice_button);
    final View voiceButtonProxy = findViewById(R.id.voice_button_proxy);

    // We only show/update the voice search icon if the search icon is enabled as well
    final SearchManager searchManager =
        (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    ComponentName globalSearchActivity = searchManager.getGlobalSearchActivity();

    ComponentName activityName = null;
    if (globalSearchActivity != null) {
        // Check if the global search activity handles voice search
        Intent intent = new Intent(RecognizerIntent.ACTION_WEB_SEARCH);
        intent.setPackage(globalSearchActivity.getPackageName());
        activityName = intent.resolveActivity(getPackageManager());
    }

    if (activityName == null) {
        // Fallback: check if an activity other than the global search activity
        // resolves this
        Intent intent = new Intent(RecognizerIntent.ACTION_WEB_SEARCH);
        activityName = intent.resolveActivity(getPackageManager());
    }

    if (searchVisible && activityName != null) {
        int coi = getCurrentOrientationIndexForGlobalIcons();
        sVoiceSearchIcon[coi] = updateButtonWithIconFromExternalActivity(
            R.id.voice_button, activityName, R.drawable.ic_home_voice_search_holo,
            TOOLBAR_VOICE_SEARCH_ICON_METADATA_NAME);
        if (sVoiceSearchIcon[coi] == null) {
            sVoiceSearchIcon[coi] = updateButtonWithIconFromExternalActivity(
                R.id.voice_button, activityName, R.drawable.ic_home_voice_search_holo,

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        TOOLBAR_ICON_METADATA_NAME);
    }
    if (voiceButtonContainer != null) voiceButtonContainer.setVisibility(View.VISIBLE);
    voiceButton.setVisibility(View.VISIBLE);
    if (voiceButtonProxy != null) {
        voiceButtonProxy.setVisibility(View.VISIBLE);
    }
    invalidatePressedFocusedStates(voiceButtonContainer, voiceButton);
    return true;
} else {
    if (voiceButtonContainer != null) voiceButtonContainer.setVisibility(View.GONE);
    voiceButton.setVisibility(View.GONE);
    if (voiceButtonProxy != null) {
        voiceButtonProxy.setVisibility(View.GONE);
    }
    return false;
}
}

private void updateVoiceSearchIcon(Drawable.ConstantState d) {
    final View voiceButtonContainer = findViewById(R.id.voice_button_container);
    final View voiceButton = findViewById(R.id.voice_button);
    updateButtonWithDrawable(R.id.voice_button, d);
    invalidatePressedFocusedStates(voiceButtonContainer, voiceButton);
}

/**
 * Sets the app market icon
 */
private void updateAppMarketIcon() {
    final View marketButton = findViewById(R.id.market_button);
    Intent intent = new
Intent(Intent.ACTION_MAIN).addCategory(Intent.CATEGORY_APP_MARKET);
    // Find the app market activity by resolving an intent.
    // (If multiple app markets are installed, it will return the ResolverActivity.)
    ComponentName activityName = intent.resolveActivity(getPackageManager());
    if (activityName != null) {
        int coi = getCurrentOrientationIndexForGlobalIcons();
        mAppMarketIntent = intent;
        sAppMarketIcon[coi] = updateTextButtonWithIconFromExternalActivity(
            R.id.market_button, activityName, R.drawable.ic_launcher_market_holo,
            TOOLBAR_ICON_METADATA_NAME);
        marketButton.setVisibility(View.VISIBLE);
    } else {
        // We should hide and disable the view so that we don't try and restore the visibility
        // of it when we swap between drag & normal states from IconDropTarget subclasses.
        marketButton.setVisibility(View.GONE);
        marketButton.setEnabled(false);
    }
}
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

private void updateAppMarketIcon(Drawable.ConstantState d) {
    // Ensure that the new drawable we are creating has the appropriate toolbar icon bounds
    Resources r = getResources();
    Drawable marketIconDrawable = d.newDrawable(r);
    int w = r.getDimensionPixelSize(R.dimen.toolbar_external_icon_width);
    int h = r.getDimensionPixelSize(R.dimen.toolbar_external_icon_height);
    marketIconDrawable.setBounds(0, 0, w, h);

    updateTextButtonWithDrawable(R.id.market_button, marketIconDrawable);
}

@Override
public boolean dispatchPopulateAccessibilityEvent(AccessibilityEvent event) {
    final boolean result = super.dispatchPopulateAccessibilityEvent(event);
    final List<CharSequence> text = event.getText();
    text.clear();
    // Populate event with a fake title based on the current state.
    if (mState == State.APPS_CUSTOMIZE) {
        text.add(getString(R.string.all_apps_button_label));
    } else {
        text.add(getString(R.string.all_apps_home_button_label));
    }
    return result;
}

/**
 * Receives notifications when system dialogs are to be closed.
 */
private class CloseSystemDialogsIntentReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        closeSystemDialogs();
    }
}

/**
 * Receives notifications whenever the appwidgets are reset.
 */
private class AppWidgetResetObserver extends ContentObserver {
    public AppWidgetResetObserver() {
        super(new Handler());
    }

    @Override
    public void onChange(boolean selfChange) {
        onAppWidgetReset();
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/**
 * If the activity is currently paused, signal that we need to run the passed Runnable
 * in onResume.
 *
 * This needs to be called from incoming places where resources might have been loaded
 * while we are paused. That is because the Configuration might be wrong
 * when we're not running, and if it comes back to what it was when we
 * were paused, we are not restarted.
 *
 * Implementation of the method from LauncherModel.Callbacks.
 *
 * @return true if we are currently paused. The caller might be able to
 * skip some work in that case since we will come back again.
 */
private boolean waitUntilResume(Runnable run, boolean deletePreviousRunnables) {
    if (mPaused) {
        Log.i(TAG, "Deferring update until onResume");
        if (deletePreviousRunnables) {
            while (mOnResumeCallbacks.remove(run)) {
            }
        }
        mOnResumeCallbacks.add(run);
        return true;
    } else {
        return false;
    }
}

private boolean waitUntilResume(Runnable run) {
    return waitUntilResume(run, false);
}

/**
 * If the activity is currently paused, signal that we need to re-run the loader
 * in onResume.
 *
 * This needs to be called from incoming places where resources might have been loaded
 * while we are paused. That is because the Configuration might be wrong
 * when we're not running, and if it comes back to what it was when we
 * were paused, we are not restarted.
 *
 * Implementation of the method from LauncherModel.Callbacks.
 *
 * @return true if we are currently paused. The caller might be able to
 * skip some work in that case since we will come back again.
 */
public boolean setLoadOnResume() {
    if (mPaused) {
        Log.i(TAG, "setLoadOnResume");
        mOnResumeNeedsLoad = true;
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        return true;
    } else {
        return false;
    }
}

/**
 * Implementation of the method from LauncherModel.Callbacks.
 */
public int getCurrentWorkspaceScreen() {
    if (mWorkspace != null) {
        return mWorkspace.getCurrentPage();
    } else {
        return SCREEN_COUNT / 2;
    }
}

/**
 * Refreshes the shortcuts shown on the workspace.
 *
 * Implementation of the method from LauncherModel.Callbacks.
 */
public void startBinding() {
    // If we're starting binding all over again, clear any bind calls we'd postponed in
    // the past (see waitUntilResume) -- we don't need them since we're starting binding
    // from scratch again
    mOnResumeCallbacks.clear();

    final Workspace workspace = mWorkspace;
    mNewShortcutAnimatePage = -1;
    mNewShortcutAnimateViews.clear();
    mWorkspace.clearDropTargets();
    int count = workspace.getChildCount();
    for (int i = 0; i < count; i++) {
        // Use removeAllViewsInLayout() to avoid an extra requestLayout() and invalidate().
        final CellLayout layoutParent = (CellLayout) workspace.getChildAt(i);
        layoutParent.removeAllViewsInLayout();
    }
    mWidgetsToAdvance.clear();
    if (mHotseat != null) {
        mHotseat.resetLayout();
    }
}

/**
 * Bind the items start-end from the list.
 *
 * Implementation of the method from LauncherModel.Callbacks.
 */
public void bindItems(final ArrayList<ItemInfo> shortcuts, final int start, final int end) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    if (waitUntilResume(new Runnable() {
        public void run() {
            bindItems(shortcuts, start, end);
        }
    })) {
        return;
    }

    // Get the list of added shortcuts and intersect them with the set of shortcuts here
    Set<String> newApps = new HashSet<String>();
    newApps = mSharedPreferences.getStringSet(InstallShortcutReceiver.NEW_APPS_LIST_KEY,
newApps);

    Workspace workspace = mWorkspace;
    for (int i = start; i < end; i++) {
        final ItemInfo item = shortcuts.get(i);

        // Short circuit if we are loading dock items for a configuration which has no dock
        if (item.container == LauncherSettings.Favorites.CONTAINER_HOTSEAT &&
            mHotseat == null) {
            continue;
        }

        switch (item.itemType) {
            case LauncherSettings.Favorites.ITEM_TYPE_APPLICATION:
            case LauncherSettings.Favorites.ITEM_TYPE_SHORTCUT:
                ShortcutInfo info = (ShortcutInfo) item;
                String uri = info.intent.toUri(0).toString();
                View shortcut = createShortcut(info);
                workspace.addInScreen(shortcut, item.container, item.screen, item.cellX,
                    item.cellY, 1, 1, false);
                boolean animateIconUp = false;
                synchronized (newApps) {
                    if (newApps.contains(uri)) {
                        animateIconUp = newApps.remove(uri);
                    }
                }
                if (animateIconUp) {
                    // Prepare the view to be animated up
                    shortcut.setAlpha(0f);
                    shortcut.setScaleX(0f);
                    shortcut.setScaleY(0f);
                    mNewShortcutAnimatePage = item.screen;
                    if (!mNewShortcutAnimateViews.contains(shortcut)) {
                        mNewShortcutAnimateViews.add(shortcut);
                    }
                }
                break;
            case LauncherSettings.Favorites.ITEM_TYPE_FOLDER:
                FolderIcon newFolder = FolderIcon.fromXml(R.layout.folder_icon, this,

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        (ViewGroup) workspace.getChildAt(workspace.getCurrentPage()),
        (FolderInfo) item, mlconCache);
workspace.addInScreen(newFolder, item.container, item.screen, item.cellX,
        item.cellY, 1, 1, false);
break;
    }
}

workspace.requestLayout();
}

/**
 * Implementation of the method from LauncherModel.Callbacks.
 */
public void bindFolders(final HashMap<Long, FolderInfo> folders) {
    if (waitUntilResume(new Runnable() {
        public void run() {
            bindFolders(folders);
        }
    })) {
        return;
    }
    sFolders.clear();
    sFolders.putAll(folders);
}

/**
 * Add the views for a widget to the workspace.
 */
* Implementation of the method from LauncherModel.Callbacks.
*/
public void bindAppWidget(final LauncherAppWidgetInfo item) {
    if (waitUntilResume(new Runnable() {
        public void run() {
            bindAppWidget(item);
        }
    })) {
        return;
    }

    final long start = DEBUG_WIDGETS ? SystemClock.uptimeMillis() : 0;
    if (DEBUG_WIDGETS) {
        Log.d(TAG, "bindAppWidget: " + item);
    }
    final Workspace workspace = mWorkspace;

    final int appWidgetId = item.appWidgetId;
    final AppWidgetProviderInfo appWidgetInfo =
mAppWidgetManager.getAppWidgetInfo(appWidgetId);
    if (DEBUG_WIDGETS) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        Log.d(TAG, "bindAppWidget: id=" + item.appWidgetId + " belongs to component " +
appWidgetInfo.provider);
    }

    item.hostView = mAppWidgetHost.createView(this, appWidgetId, appWidgetInfo);

    item.hostView.setTag(item);
    item.onBindAppWidget(this);

    workspace.addInScreen(item.hostView, item.container, item.screen, item.cellX,
        item.cellY, item.spanX, item.spanY, false);
    addWidgetToAutoAdvanceIfNeeded(item.hostView, appWidgetInfo);

    workspace.requestLayout();

    if (DEBUG_WIDGETS) {
        Log.d(TAG, "bound widget id="+item.appWidgetId+" in "
            + (SystemClock.uptimeMillis()-start) + "ms");
    }
}

public void onPageBoundSynchronously(int page) {
    mSynchronouslyBoundPages.add(page);
}

/**
 * Callback saying that there aren't any more items to bind.
 */
* Implementation of the method from LauncherModel.Callbacks.
*/
public void finishBindingItems() {
    if (waitUntilResume(new Runnable() {
        public void run() {
            finishBindingItems();
        }
    })) {
        return;
    }
    if (mSavedState != null) {
        if (!mWorkspace.hasFocus()) {
            mWorkspace.getChildAt(mWorkspace.getCurrentPage()).requestFocus();
        }
        mSavedState = null;
    }

    mWorkspace.restoreInstanceStateForRemainingPages();

    // If we received the result of any pending adds while the loader was running (e.g. the
    // widget configuration forced an orientation change), process them now.
    for (int i = 0; i < sPendingAddList.size(); i++) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        completeAdd(sPendingAddList.get(i));
    }
    sPendingAddList.clear();

    // Update the market app icon as necessary (the other icons will be managed in response
to
    // package changes in bindSearchablesChanged()
    updateAppMarketIcon();

    // Animate up any icons as necessary
    if (mVisible || mWorkspaceLoading) {
        Runnable newAppsRunnable = new Runnable() {
            @Override
            public void run() {
                runNewAppsAnimation(false);
            }
        };

        boolean willSnapPage = mNewShortcutAnimatePage > -1 &&
            mNewShortcutAnimatePage != mWorkspace.getCurrentPage();
        if (canRunNewAppsAnimation()) {
            // If the user has not interacted recently, then either snap to the new page to show
            // the new-apps animation or just run them if they are to appear on the current page
            if (willSnapPage) {
                mWorkspace.snapToPage(mNewShortcutAnimatePage, newAppsRunnable);
            } else {
                runNewAppsAnimation(false);
            }
        } else {
            // If the user has interacted recently, then just add the items in place if they
            // are on another page (or just normally if they are added to the current page)
            runNewAppsAnimation(willSnapPage);
        }
    }

    mWorkspaceLoading = false;
}

private boolean canRunNewAppsAnimation() {
    long diff = System.currentTimeMillis() - mDragController.getLastGestureUpTime();
    return diff > (NEW_APPS_ANIMATION_INACTIVE_TIMEOUT_SECONDS * 1000);
}

/**
 * Runs a new animation that scales up icons that were added while Launcher was in the
 * background.
 *
 * @param immediate whether to run the animation or show the results immediately
 */
private void runNewAppsAnimation(boolean immediate) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

AnimatorSet anim = LauncherAnimUtils.createAnimatorSet();
Collection<Animator> bounceAnims = new ArrayList<Animator>();

// Order these new views spatially so that they animate in order
Collections.sort(mNewShortcutAnimateViews, new Comparator<View>() {
    @Override
    public int compare(View a, View b) {
        CellLayout.LayoutParams alp = (CellLayout.LayoutParams) a.getLayoutParams();
        CellLayout.LayoutParams blp = (CellLayout.LayoutParams) b.getLayoutParams();
        int cellCountX = LauncherModel.getCellCountX();
        return (alp.cellY * cellCountX + alp.cellX) - (blp.cellY * cellCountX + blp.cellX);
    }
});

// Animate each of the views in place (or show them immediately if requested)
if (immediate) {
    for (View v : mNewShortcutAnimateViews) {
        v.setAlpha(1f);
        v.setScaleX(1f);
        v.setScaleY(1f);
    }
} else {
    for (int i = 0; i < mNewShortcutAnimateViews.size(); ++i) {
        View v = mNewShortcutAnimateViews.get(i);
        ValueAnimator bounceAnim = LauncherAnimUtils.ofPropertyValuesHolder(v,
            PropertyValuesHolder.ofFloat("alpha", 1f),
            PropertyValuesHolder.ofFloat("scaleX", 1f),
            PropertyValuesHolder.ofFloat("scaleY", 1f));

        bounceAnim.setDuration(InstallShortcutReceiver.NEW_SHORTCUT_BOUNCE_DURATION);
        bounceAnim.setStartDelay(i *
InstallShortcutReceiver.NEW_SHORTCUT_STAGGER_DELAY);
        bounceAnim.setInterpolator(new SmoothPagedView.OvershootInterpolator());
        bounceAnims.add(bounceAnim);
    }
    anim.playTogether(bounceAnims);
    anim.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationEnd(Animator animation) {
            if (mWorkspace != null) {
                mWorkspace.postDelayed(mBuildLayersRunnable, 500);
            }
        }
    });
    anim.start();
}

// Clean up
mNewShortcutAnimatePage = -1;
mNewShortcutAnimateViews.clear();

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

new Thread("clearNewAppsThread") {
    public void run() {
        mSharedPreferences.edit()
            .putInt(InstallShortcutReceiver.NEW_APPS_PAGE_KEY, -1)
            .putStringSet(InstallShortcutReceiver.NEW_APPS_LIST_KEY, null)
            .commit();
    }
}.start();
}

@Override
public void bindSearchablesChanged() {
    boolean searchVisible = updateGlobalSearchIcon();
    boolean voiceVisible = updateVoiceSearchIcon(searchVisible);
    if (mSearchDropTargetBar != null) {
        mSearchDropTargetBar.onSearchPackagesChanged(searchVisible, voiceVisible);
    }
}

/**
 * Add the icons for all apps.
 *
 * Implementation of the method from LauncherModel.Callbacks.
 */
public void bindAllApplications(final ArrayList<ApplicationInfo> apps) {
    Runnable setAllAppsRunnable = new Runnable() {
        public void run() {
            if (mAppsCustomizeContent != null) {
                mAppsCustomizeContent.setApps(apps);
            }
        }
    };

    // Remove the progress bar entirely; we could also make it GONE
    // but better to remove it since we know it's not going to be used
    View progressBar = mAppsCustomizeTabHost.
        findViewById(R.id.apps_customize_progress_bar);
    if (progressBar != null) {
        ((ViewGroup)progressBar.getParent()).removeView(progressBar);

        // We just post the call to setApps so the user sees the progress bar
        // disappear-- otherwise, it just looks like the progress bar froze
        // which doesn't look great
        mAppsCustomizeTabHost.post(setAllAppsRunnable);
    } else {
        // If we did not initialize the spinner in onCreate, then we can directly set the
        // list of applications without waiting for any progress bars views to be hidden.
        setAllAppsRunnable.run();
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/**
 * A package was installed.
 *
 * Implementation of the method from LauncherModel.Callbacks.
 */
public void bindAppsAdded(final ArrayList<ApplicationInfo> apps) {
    if (waitUntilResume(new Runnable() {
        public void run() {
            bindAppsAdded(apps);
        }
    })) {
        return;
    }
}

```

```

    if (mAppsCustomizeContent != null) {
        mAppsCustomizeContent.addApps(apps);
    }
}

```

```

/**
 * A package was updated.
 *
 * Implementation of the method from LauncherModel.Callbacks.
 */
public void bindAppsUpdated(final ArrayList<ApplicationInfo> apps) {
    if (waitUntilResume(new Runnable() {
        public void run() {
            bindAppsUpdated(apps);
        }
    })) {
        return;
    }
}

```

```

    if (mWorkspace != null) {
        mWorkspace.updateShortcuts(apps);
    }
}

```

```

    if (mAppsCustomizeContent != null) {
        mAppsCustomizeContent.updateApps(apps);
    }
}

```

```

/**
 * A package was uninstalled. We take both the super set of packageNames
 * in addition to specific applications to remove, the reason being that
 * this can be called when a package is updated as well. In that scenario,
 * we only remove specific components from the workspace, where as
 * package-removal should clear all items by package name.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

*
* Implementation of the method from LauncherModel.Callbacks.
*/
public void bindComponentsRemoved(final ArrayList<String> packageNames,
                                final ArrayList<ApplicationInfo> applInfos,
                                final boolean matchPackageNamesOnly) {
    if (waitUntilResume(new Runnable() {
        public void run() {
            bindComponentsRemoved(packageNames, applInfos, matchPackageNamesOnly);
        }
    })) {
        return;
    }

    if (matchPackageNamesOnly) {
        mWorkspace.removeItemsByPackageName(packageNames);
    } else {
        mWorkspace.removeItemsByApplicationInfo(applInfos);
    }

    if (mAppsCustomizeContent != null) {
        mAppsCustomizeContent.removeApps(applInfos);
    }

    // Notify the drag controller
    mDragController.onAppsRemoved(applInfos, this);
}

/**
 * A number of packages were updated.
 */

private ArrayList<Object> mWidgetsAndShortcuts;
private Runnable mBindPackagesUpdatedRunnable = new Runnable() {
    public void run() {
        bindPackagesUpdated(mWidgetsAndShortcuts);
        mWidgetsAndShortcuts = null;
    }
};

public void bindPackagesUpdated(final ArrayList<Object> widgetsAndShortcuts) {
    if (waitUntilResume(mBindPackagesUpdatedRunnable, true)) {
        mWidgetsAndShortcuts = widgetsAndShortcuts;
        return;
    }

    if (mAppsCustomizeContent != null) {
        mAppsCustomizeContent.onPackagesUpdated(widgetsAndShortcuts);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

private int mapConfigurationOriActivityInfoOri(int configOri) {
    final Display d = getWindowManager().getDefaultDisplay();
    int naturalOri = Configuration.ORIENTATION_LANDSCAPE;
    switch (d.getRotation()) {
        case Surface.ROTATION_0:
        case Surface.ROTATION_180:
            // We are currently in the same basic orientation as the natural orientation
            naturalOri = configOri;
            break;
        case Surface.ROTATION_90:
        case Surface.ROTATION_270:
            // We are currently in the other basic orientation to the natural orientation
            naturalOri = (configOri == Configuration.ORIENTATION_LANDSCAPE) ?
                Configuration.ORIENTATION_PORTRAIT :
                Configuration.ORIENTATION_LANDSCAPE;
            break;
    }

    int[] oriMap = {
        ActivityInfo.SCREEN_ORIENTATION_PORTRAIT,
        ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE,
        ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT,
        ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDSCAPE
    };
    // Since the map starts at portrait, we need to offset if this device's natural orientation
    // is landscape.
    int indexOffset = 0;
    if (naturalOri == Configuration.ORIENTATION_LANDSCAPE) {
        indexOffset = 1;
    }
    return oriMap[(d.getRotation() + indexOffset) % 4];
}

public boolean isRotationEnabled() {
    boolean enableRotation = sForceEnableRotation ||
        getResources().getBoolean(R.bool.allow_rotation);
    return enableRotation;
}

public void lockScreenOrientation() {
    if (isRotationEnabled()) {
        setRequestedOrientation(mapConfigurationOriActivityInfoOri(getResources()
            .getConfiguration().orientation));
    }
}

public void unlockScreenOrientation(boolean immediate) {
    if (isRotationEnabled()) {
        if (immediate) {
            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
        } else {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        mHandler.postDelayed(new Runnable() {
            public void run() {

setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
            }
        }, mRestoreScreenOrientationDelay);
    }
}

/* Cling related */
private boolean isClingsEnabled() {
    // disable clings when running in a test harness
    if(ActivityResult.isRunningInTestHarness()) return false;

    // Restricted secondary users (child mode) will potentially have very few apps
    // seeded when they start up for the first time. Clings won't work well with that
    boolean supportsLimitedUsers =
        android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.JELLY_BEAN_MR2;
    Account[] accounts = AccountManager.get(this).getAccounts();
    if (supportsLimitedUsers && accounts.length == 0) {
        UserManager um = (UserManager) getSystemService(Context.USER_SERVICE);
        Bundle restrictions = um.getUserRestrictions();
        if (restrictions.getBoolean(UserManager.DISALLOW_MODIFY_ACCOUNTS, false)) {
            return false;
        }
    }
    return true;
}

private Cling initCling(int clingId, int[] positionData, boolean animate, int delay) {
    final Cling cling = (Cling) findViewById(clingId);
    if (cling != null) {
        cling.init(this, positionData);
        cling.setVisibility(View.VISIBLE);
        cling.setLayerType(View.LAYER_TYPE_HARDWARE, null);
        if (animate) {
            cling.buildLayer();
            cling.setAlpha(0f);
            cling.animate()
                .alpha(1f)
                .setInterpolator(new AccelerateInterpolator())
                .setDuration(SHOW_CLING_DURATION)
                .setStartDelay(delay)
                .start();
        } else {
            cling.setAlpha(1f);
        }
        cling.setFocusableInTouchMode(true);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        cling.post(new Runnable() {
            public void run() {
                cling.setFocusable(true);
                cling.requestFocus();
            }
        });
        mHideFromAccessibilityHelper.setImportantForAccessibilityToNo(
            mDragLayer, clingId == R.id.all_apps_cling);
    }
    return cling;
}

private void dismissCling(final Cling cling, final String flag, int duration) {
    // To catch cases where siblings of top-level views are made invisible, just check whether
    // the cling is directly set to GONE before dismissing it.
    if (cling != null && cling.getVisibility() != View.GONE) {
        ObjectAnimator anim = LauncherAnimUtils.ofFloat(cling, "alpha", 0f);
        anim.setDuration(duration);
        anim.addListener(new AnimatorListenerAdapter() {
            public void onAnimationEnd(Animator animation) {
                cling.setVisibility(View.GONE);
                cling.cleanup();
                // We should update the shared preferences on a background thread
                new Thread("dismissClingThread") {
                    public void run() {
                        SharedPreferences.Editor editor = mSharedPrefs.edit();
                        editor.putBoolean(flag, true);
                        editor.commit();
                    }
                }.start();
            }
        });
        anim.start();
        mHideFromAccessibilityHelper.restoreImportantForAccessibility(mDragLayer);
    }
}

private void removeCling(int id) {
    final View cling = findViewById(id);
    if (cling != null) {
        final ViewGroup parent = (ViewGroup) cling.getParent();
        parent.post(new Runnable() {
            @Override
            public void run() {
                parent.removeView(cling);
            }
        });
        mHideFromAccessibilityHelper.restoreImportantForAccessibility(mDragLayer);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

private boolean skipCustomClingIfNoAccounts() {
    Cling cling = (Cling) findViewById(R.id.workspace_cling);
    boolean customCling = cling.getDrawableIdentifier().equals("workspace_custom");
    if (customCling) {
        AccountManager am = AccountManager.get(this);
        Account[] accounts = am.getAccountsByType("com.google");
        return accounts.length == 0;
    }
    return false;
}

public void showFirstRunWorkspaceCling() {
    // Enable the clings only if they have not been dismissed before
    if (isClingsEnabled() &&
        !mSharedPreferences.getBoolean(Cling.WORKSPACE_CLING_DISMISSED_KEY, false)
    &&
        !skipCustomClingIfNoAccounts() ) {
        // If we're not using the default workspace layout, replace workspace cling
        // with a custom workspace cling (usually specified in an overlay)
        // For now, only do this on tablets
        if (mSharedPreferences.getInt(LauncherProvider.DEFAULT_WORKSPACE_RESOURCE_ID,
0) != 0 &&
            getResources().getBoolean(R.bool.config_useCustomClings)) {
            // Use a custom cling
            View cling = findViewById(R.id.workspace_cling);
            ViewGroup clingParent = (ViewGroup) cling.getParent();
            int clingIndex = clingParent.indexOfChild(cling);
            clingParent.removeViewAt(clingIndex);
            View customCling = mInflater.inflate(R.layout.custom_workspace_cling, clingParent,
false);
            clingParent.addView(customCling, clingIndex);
            customCling.setId(R.id.workspace_cling);
        }
        initCling(R.id.workspace_cling, null, false, 0);
    } else {
        removeCling(R.id.workspace_cling);
    }
}

public void showFirstRunAllAppsCling(int[] position) {
    // Enable the clings only if they have not been dismissed before
    if (isClingsEnabled() &&
        !mSharedPreferences.getBoolean(Cling.ALLAPPS_CLING_DISMISSED_KEY, false)) {
        initCling(R.id.all_apps_cling, position, true, 0);
    } else {
        removeCling(R.id.all_apps_cling);
    }
}

public void showFirstRunFoldersCling() {
    // Enable the clings only if they have not been dismissed before

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    if (isClingsEnabled() &&
        !mSharedPreferences.getBoolean(Cling.FOLDER_CLING_DISMISSED_KEY, false)) {
        return initCling(R.id.folder_cling, null, true, 0);
    } else {
        removeCling(R.id.folder_cling);
        return null;
    }
}

public boolean isFolderClingVisible() {
    Cling cling = (Cling) findViewById(R.id.folder_cling);
    if (cling != null) {
        return cling.getVisibility() == View.VISIBLE;
    }
    return false;
}

public void dismissWorkspaceCling(View v) {
    Cling cling = (Cling) findViewById(R.id.workspace_cling);
    dismissCling(cling, Cling.WORKSPACE_CLING_DISMISSED_KEY,
DISMISS_CLING_DURATION);
}

public void dismissAllAppsCling(View v) {
    Cling cling = (Cling) findViewById(R.id.all_apps_cling);
    dismissCling(cling, Cling.ALLAPPS_CLING_DISMISSED_KEY,
DISMISS_CLING_DURATION);
}

public void dismissFolderCling(View v) {
    Cling cling = (Cling) findViewById(R.id.folder_cling);
    dismissCling(cling, Cling.FOLDER_CLING_DISMISSED_KEY,
DISMISS_CLING_DURATION);
}

/**
 * Prints out out state for debugging.
 */
public void dumpState() {
    Log.d(TAG, "BEGIN launcher2 dump state for launcher " + this);
    Log.d(TAG, "mSavedState=" + mSavedState);
    Log.d(TAG, "mWorkspaceLoading=" + mWorkspaceLoading);
    Log.d(TAG, "mRestoring=" + mRestoring);
    Log.d(TAG, "mWaitingForResult=" + mWaitingForResult);
    Log.d(TAG, "mSavedInstanceState=" + mSavedInstanceState);
    Log.d(TAG, "sFolders.size=" + sFolders.size());
    mModel.dumpState();

    if (mAppsCustomizeContent != null) {
        mAppsCustomizeContent.dumpState();
    }
    Log.d(TAG, "END launcher2 dump state");
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

@Override
public void dump(String prefix, FileDescriptor fd, PrintWriter writer, String[] args) {
    super.dump(prefix, fd, writer, args);
    writer.println(" ");
    writer.println("Debug logs: ");
    for (int i = 0; i < sDumpLogs.size(); i++) {
        writer.println(" " + sDumpLogs.get(i));
    }
}

public static void dumpDebugLogsToConsole() {
    Log.d(TAG, "");
    Log.d(TAG, "*****");
    Log.d(TAG, "Launcher debug logs: ");
    for (int i = 0; i < sDumpLogs.size(); i++) {
        Log.d(TAG, " " + sDumpLogs.get(i));
    }
    Log.d(TAG, "*****");
    Log.d(TAG, "");
}
}

interface LauncherTransitionable {
    View getContent();
    void onLauncherTransitionPrepare(Launcher l, boolean animated, boolean toWorkspace);
    void onLauncherTransitionStart(Launcher l, boolean animated, boolean toWorkspace);
    void onLauncherTransitionStep(Launcher l, float t);
    void onLauncherTransitionEnd(Launcher l, boolean animated, boolean toWorkspace);
}
}
xix android-platform_frameworks_base-
57ea96a\core\java\android\appwidget\AppDataWidgetHost.java

/**
 * Start receiving onAppWidgetChanged calls for your AppWidgets. Call this when your
 * activity
 * becomes visible, i.e. from onStart() in your Activity.
 */
public void startListening() {
    int[] updatedIds;
    ArrayList<RemoteViews> updatedViews = new ArrayList<RemoteViews>();

    try {
        if (mPackageName == null) {
            mPackageName = mContext.getPackageName();
        }
        updatedIds = sService.startListening(mCallbacks, mPackageName, mHostId,
        updatedViews);
    }
    catch (RemoteException e) {
        throw new RuntimeException("system server dead?", e);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

finalint N = updatedIds.length;
for (int i=0; i<N; i++) {
    updateAppWidgetView(updatedIds[i], updatedViews.get(i));
}
}

xx    \platform_dalvik-master\vm\interp\Interp.cpp

/*
 * Main interpreter loop entry point.
 *
 * This begins executing code at the start of "method". On exit, "pResult"
 * holds the return value of the method (or, if "method" returns NULL, it
 * holds an undefined value).
 *
 * The interpreted stack frame, which holds the method arguments, has
 * already been set up.
 */
void dvmInterpret(Thread* self, const Method* method, JValue* pResult)
{
    typedef void (*Interpreter)(Thread*);
    Interpreter stdInterp;
    if (gDvm.executionMode == kExecutionModeInterpFast)
        stdInterp = dvmMterpStd;
    #if defined(WITH_JIT)
    elseif (gDvm.executionMode == kExecutionModeJit ||
            gDvm.executionMode == kExecutionModeNcgO0 ||
            gDvm.executionMode == kExecutionModeNcgO1)
        stdInterp = dvmMterpStd;
    #endif
    else
        stdInterp = dvmInterpretPortable;

    // Call the interpreter
    (*stdInterp)(self);

```

```

xxi    \platform_dalvik-master\vm\mterp\out\InterpC-portable.cpp

```

```

/* File: portable/entry.cpp */
/*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* Main interpreter loop.
*
* This was written with an ARM implementation in mind.
*/
void dvmInterpretPortable(Thread* self)
{
#if defined(EASY_GDB)
StackSaveArea* debugSaveArea = SAVEAREA_FROM_FP(self->interpSave.curFrame);
#endif
DvmDex* methodClassDex;    // curMethod->clazz->pDvmDex
JValueretval;

/* core state */
const Method* curMethod;    // method we're interpreting
const u2* pc;                // program counter
u4* fp;                      // frame pointer
u2inst;                      // current instruction
/* instruction decoding */
u4 ref;                      // 16 or 32-bit quantity fetched directly
u2 vsrc1, vsrc2, vdst;       // usually used for register indexes
/* method call setup */
const Method* methodToCall;
bool methodCallRange;

/* static computed goto table */
    DEFINE_GOTO_TABLE(handlerTable);

/* copy state in */
curMethod = self->interpSave.method;
pc = self->interpSave.pc;
fp = self->interpSave.curFrame;
retval = self->interpSave.retval; /* only need for kInterpEntryReturn? */

methodClassDex = curMethod->clazz->pDvmDex;

    LOGVV("threadid=%d: %s.%s pc=%#x fp=%p",
        self->threadId, curMethod->clazz->descriptor, curMethod->name,
        pc - curMethod->insns, fp);

/*
* Handle any ongoing profiling and prep for debugging.
*/
if (self->interpBreak.ctl.subMode != 0) {
    TRACE_METHOD_ENTER(self, curMethod);
    self->debugIsMethodEntry = true; // Always true on startup
}
/*
* DEBUG: scramble this to ensure we're not relying on it.
*/
methodToCall = (const Method*) -1;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

#if 0
if (self->debugIsMethodEntry) {
    ILOGD("|-- Now interpreting %s.%s", curMethod->clazz->descriptor,
    curMethod->name);
    DUMP_REGS(curMethod, self->interpSave.curFrame, false);
}
#endif

    FINISH(0);          /* fetch and execute first instruction */

```

```

xxii      \platform_dalvik-master\vm\interp\out\InterpC-portable.cpp

```

```

/*--- start of opcodes ---*/

/* File: c/OP_CONST_CLASS.cpp */
HANDLE_OPCODE(OP_CONST_CLASS/*vAA, class@BBBB*/)
{
    ClassObject* clazz;

    vdst = INST_AA(inst);
    ref = FETCH(1);
    ILOGV("|const-class v%d class@0x%04x", vdst, ref);
    clazz = dvmDexGetResolvedClass(methodClassDex, ref);
    if (clazz == NULL) {
        EXPORT_PC();
        clazz = dvmResolveClass(curMethod->clazz, ref, true);
        if (clazz == NULL)
            GOTO_exceptionThrown();
    }
    SET_REGISTER(vdst, (u4) clazz);
}
    FINISH(2);
OP_END

GOTO_TARGET(invokerStatic, boolmethodCallRange)
    EXPORT_PC();

    vsrc1 = INST_AA(inst); /* AA (count) or BA (count + arg 5) */
    ref = FETCH(1);        /* method ref */
    vdst = FETCH(2);        /* 4 regs -or- first reg */

if (methodCallRange)
    ILOGV("|invoke-static-range args=%d @0x%04x {regs=v%d-v%d}",
    vsrc1, ref, vdst, vdst+vsrc1-1);
else

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```
ILOGV("|invoke-static args=%d @0x%04x {regs=0x%04x %x}",
      vsrc1 >> 4, ref, vdst, vsrc1 & 0x0f);
```

```
methodToCall = dvmDexGetResolvedMethod(methodClassDex, ref);
if (methodToCall == NULL) {
methodToCall = dvmResolveMethod(curMethod->clazz, ref, METHOD_STATIC);
if (methodToCall == NULL) {
    ILOGV("+ unknown method");
GOTO_exceptionThrown();
}
}
```

```
xxiii      \platform_dalvik-master\vm\oo\Resolve.cpp
```

```
/*
 * Find the class corresponding to "classIdx", which maps to a class name
 * string. It might be in the same DEX file as "referrer", in a different
 * DEX file, generated by a class loader, or generated by the VM (e.g.
 * array classes).
 *
 * Because the DexTypeIdx is associated with the referring class' DEX file,
 * we may have to resolve the same class more than once if it's referred
 * to from classes in multiple DEX files. This is a necessary property for
 * DEX files associated with different class loaders.
 *
 * We cache a copy of the lookup in the DexFile's "resolved class" table,
 * so future references to "classIdx" are faster.
 *
 * Note that "referrer" may be in the process of being linked.
 *
 * Traditional VMs might do access checks here, but in Dalvik the class
 * "constant pool" is shared between all classes in the DEX file. We rely
 * on the verifier to do the checks for us.
 *
 * Does not initialize the class.
 *
 * "fromUnverifiedConstant" should only be set if this call is the direct
 * result of executing a "const-class" or "instance-of" instruction, which
 * use class constants not resolved by the bytecode verifier.
 *
 * Returns NULL with an exception raised on failure.
 */
```

```
ClassObject* dvmResolveClass(constClassObject* referrer, u4classIdx,
boolfromUnverifiedConstant)
{
DvmDex* pDvmDex = referrer->pDvmDex;
ClassObject* resClass;
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

constchar* className;

/*
 * Check the table first -- this gets called from the other "resolve"
 * methods.
 */
resClass = dvmDexGetResolvedClass(pDvmDex, classIdx);
if (resClass != NULL)
return resClass;

    LOGVV("--- resolving class %u (referrer=%s cl=%p)",
    classIdx, referrer->descriptor, referrer->classLoader);

/*
 * Class hasn't been loaded yet, or is in the process of being loaded
 * and initialized now. Try to get a copy. If we find one, put the
 * pointer in the DexTypeIdx. There isn't a race condition here --
 * 32-bit writes are guaranteed atomic on all target platforms. Worst
 * case we have two threads storing the same value.
 *
 * If this is an array class, we'll generate it here.
 */
className = dexStringByTypeIdx(pDvmDex->pDexFile, classIdx);
if (className[0] != '\0' && className[1] == '\0') {
/* primitive type */
resClass = dvmFindPrimitiveClass(className[0]);
} else {
resClass = dvmFindClassNoInit(className, referrer->classLoader);
}

if (resClass != NULL) {
/*
 * If the referrer was pre-verified, the resolved class must come
 * from the same DEX or from a bootstrap class. The pre-verifier
 * makes assumptions that could be invalidated by a wacky class
 * loader. (See the notes at the top of oo/Class.c.)
 *
 * The verifier does not fail a class for using a const-class
 * or instance-of instruction referring to an unresolveable class,
 * because the result of the instruction is simply a Class object
 * or boolean -- there's no need to resolve the class object during
 * verification. Instance field and virtual method accesses can
 * break dangerously if we get the wrong class, but const-class and
 * instance-of are only interesting at execution time. So, if we
 * we got here as part of executing one of the "unverified class"
 * instructions, we skip the additional check.
 *
 * Ditto for class references from annotations and exception
 * handler lists.
 */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

if (!fromUnverifiedConstant&&
    IS_CLASS_FLAG_SET(referrer, CLASS_ISPREVERIFIED))
{
    ClassObject* resClassCheck = resClass;
    if (dvmIsArrayClass(resClassCheck))
        resClassCheck = resClassCheck->elementClass;

    if (referrer->pDvmDex != resClassCheck->pDvmDex&&
        resClassCheck->classLoader != NULL)
    {
        ALOGW("Class resolved by unexpected DEX:"
            " %s(%p):%p ref [%s] %s(%p):%p",
            referrer->descriptor, referrer->classLoader,
            referrer->pDvmDex,
            resClass->descriptor, resClassCheck->descriptor,
            resClassCheck->classLoader, resClassCheck->pDvmDex);
        ALOGW("(%s had used a different %s during pre-verification)",
            referrer->descriptor, resClass->descriptor);
        dvmThrowIllegalAccessError(
            "Class ref in pre-verified class resolved to unexpected "
            "implementation");
        return NULL;
    }
}

LOGVV("##### +ResolveClass(%s): referrer=%s dex=%p ldr=%p ref=%d",
    resClass->descriptor, referrer->descriptor, referrer->pDvmDex,
    referrer->classLoader, classIdx);

/*
 * Add what we found to the list so we can skip the class search
 * next time through.
 *
 * TODO: should we be doing this when fromUnverifiedConstant==true?
 * (see comments at top of oo/Class.c)
 */
dvmDexSetResolvedClass(pDvmDex, classIdx, resClass);
} else {
/* not found, exception should be raised */
    LOGVV("Class not found: %s",
        dexStringByTypeIdx(pDvmDex->pDexFile, classIdx));
    assert(dvmCheckException(dvmThreadSelf()));
}

return resClass;
}

/*
 * Find the method corresponding to "methodRef".

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

*
* We use "referrer" to find the DexFile with the constant pool that
* "methodRef" is an index into. We also use its class loader. The method
* being resolved may very well be in a different DEX file.
*
* If this is a static method, we ensure that the method's class is
* initialized.
*/
Method* dvmResolveMethod(constClassObject* referrer, u4methodIdx,
MethodType methodType)
{
DvmDex* pDvmDex = referrer->pDvmDex;
ClassObject* resClass;
constDexMethodId* pMethodId;
Method* resMethod;

    assert(methodType != METHOD_INTERFACE);

    LOGVV("--- resolving method %u (referrer=%s)", methodIdx,
        referrer->descriptor);
    pMethodId = dexGetMethodId(pDvmDex->pDexFile, methodIdx);

    resClass = dvmResolveClass(referrer, pMethodId->classIdx, false);
    if (resClass == NULL) {
        /* can't find the class that the method is a part of */
        assert(dvmCheckException(dvmThreadSelf()));
        return NULL;
    }
    if (dvmIsInterfaceClass(resClass)) {
        /* method is part of an interface */
        dvmThrowIncompatibleClassChangeErrorWithClassMessage(
            resClass->descriptor);
        return NULL;
    }

    constchar* name = dexStringByIdx(pDvmDex->pDexFile, pMethodId->nameIdx);
    DexProto proto;
    dexProtoSetFromMethodId(&proto, pDvmDex->pDexFile, pMethodId);

    /*
     * We need to chase up the class hierarchy to find methods defined
     * in super-classes. (We only want to check the current class
     * if we're looking for a constructor; since DIRECT calls are only
     * for constructors and private methods, we don't want to walk up.)
     */
    if (methodType == METHOD_DIRECT) {
        resMethod = dvmFindDirectMethod(resClass, name, &proto);
    } elseif (methodType == METHOD_STATIC) {
        resMethod = dvmFindDirectMethodHier(resClass, name, &proto);
    } else {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

resMethod = dvmFindVirtualMethodHier(resClass, name, &proto);
}

if (resMethod == NULL) {
std::string msg;
msg += resClass->descriptor;
msg += ".";
msg += name;
dvmThrowNoSuchMethodError(msg.c_str());
return NULL;
}

LOGVV("--- found method %d (%s.%s)",
methodIdx, resClass->descriptor, resMethod->name);

/* see if this is a pure-abstract method */
if (dvmlsAbstractMethod(resMethod) && !dvmlsAbstractClass(resClass)) {
dvmThrowAbstractMethodError(name);
return NULL;
}

/*
 * If we're the first to resolve this class, we need to initialize
 * it now. Only necessary for METHOD_STATIC.
 */
if (methodType == METHOD_STATIC) {
if (!dvmlsClassInitialized(resMethod->clazz) &&
!dvmlInitClass(resMethod->clazz))
{
assert(dvmCheckException(dvmThreadSelf()));
return NULL;
} else {
assert(!dvmCheckException(dvmThreadSelf()));
}
} else {
/*
 * Edge case: if the <clinit> for a class creates an instance
 * of itself, we will call <init> on a class that is still being
 * initialized by us.
 */
assert(dvmlsClassInitialized(resMethod->clazz) ||
dvmlsClassInitializing(resMethod->clazz));
}

/*
 * If the class has been initialized, add a pointer to our data structure
 * so we don't have to jump through the hoops again. If this is a
 * static method and the defining class is still initializing (i.e. this
 * thread is executing <clinit>), don't do the store, otherwise other
 * threads could call the method without waiting for class init to finish.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    */
    if (methodType == METHOD_STATIC && !dvmIsClassInitialized(resMethod->clazz))
    {
        LOGVV("--- not caching resolved method %s.%s (class init=%d/%d)",
        resMethod->clazz->descriptor, resMethod->name,
        dvmIsClassInitializing(resMethod->clazz),
        dvmIsClassInitialized(resMethod->clazz));
    } else {
        dvmDexSetResolvedMethod(pDvmDex, methodIdx, resMethod);
    }

    return resMethod;
}

```

xxiv \platform\_dalvik-master\vm\DvmDex.h

```

/*
 * Return the requested item if it has been resolved, or NULL if it hasn't.
 */
INLINE structStringObject* dvmDexGetResolvedString(constDvmDex* pDvmDex,
u4stringIdx)
{
    assert(stringIdx < pDvmDex->pHeader->stringIdsSize);
    return pDvmDex->pResStrings[stringIdx];
}
INLINE structClassObject* dvmDexGetResolvedClass(constDvmDex* pDvmDex,
u4classIdx)
{
    assert(classIdx < pDvmDex->pHeader->typeIdsSize);
    return pDvmDex->pResClasses[classIdx];
}
INLINE structMethod* dvmDexGetResolvedMethod(constDvmDex* pDvmDex,
u4methodIdx)
{
    assert(methodIdx < pDvmDex->pHeader->methodIdsSize);
    return pDvmDex->pResMethods[methodIdx];
}
INLINE structField* dvmDexGetResolvedField(constDvmDex* pDvmDex,
u4fieldIdx)
{
    assert(fieldIdx < pDvmDex->pHeader->fieldIdsSize);
    return pDvmDex->pResFields[fieldIdx];
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

xxv \platform_dalvik-master\vm\mterp\Mterp.cpp
/*
 * "Mterp entry point.
 */
void dvmMterpStd(Thread* self)
{
    /* configure mterp items */
    self->interpSave.methodClassDex = self->interpSave.method->clazz->pDvmDex;

    IF_LOGVV() {
        char* desc = dexProtoCopyMethodDescriptor(
            &self->interpSave.method->prototype);
        LOGVV("mterp threadid=%d : %s.%s %s",
            dvmThreadSelf()->threadId,
            self->interpSave.method->clazz->descriptor,
            self->interpSave.method->name,
            desc);
        free(desc);
    }
    //ALOGI("self is %p, pc=%p, fp=%p", self, self->interpSave.pc,
    //    self->interpSave.curFrame);
    //ALOGI("first instruction is 0x%04x", self->interpSave.pc[0]);

    /*
     * Handle any ongoing profiling and prep for debugging
     */
    if (self->interpBreak.ctl.subMode != 0) {
        TRACE_METHOD_ENTER(self, self->interpSave.method);
        self->debugIsMethodEntry = true; // Always true on startup
    }

    dvmMterpStdRun(self);

#ifdef LOG_INSTR
    ALOGD("|-- Leaving interpreter loop");
#endif
}

```

```

xxvi \platform_dalvik-master\vm\mterp\out\InterpAsm-armv5te-vfp.S

```

```

/*
 * Interpreter entry point.
 */

/*
 * We don't have formal stack frames, so gdb scans upward in the code
 * to find the start of the function (a label with the %function type),

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* and then looks at the next few instructions to figure out what
* got pushed onto the stack. From this it figures out how to restore
* the registers, including PC, for the previous stack frame. If gdb
* sees a non-function label, it stops scanning, so either we need to
* have nothing but assembler-local labels between the entry point and
* the break, or we need to fake it out.
*
* When this is defined, we add some stuff to make gdb less confused.
*/
#define ASSIST_DEBUGGER 1

    .text
    .align 2
    .global dvmMterpStdRun
    .type dvmMterpStdRun, %function

/*
* On entry:
* r0 Thread* self
*
* The return comes via a call to dvmMterpStdBail().
*/
dvmMterpStdRun:
#define MTERP_ENTRY1 \
    .save {r4-r10,fp,lr}; \
    stmfd sp!, {r4-r10,fp,lr} @ save 9 regs
#define MTERP_ENTRY2 \
    .pad #4; \
    sub sp, sp, #4 @ align 64

    .fnstart
    MTERP_ENTRY1
    MTERP_ENTRY2

/* save stack pointer, add magic word for debuggerd */
str sp, [r0, #offThread_bailPtr] @ save SP for eventual return

/* set up "named" registers, figure out entry point */
mov rSELF, r0 @ set rSELF
LOAD_PC_FP_FROM_SELF() @ load rPC and rFP from "thread"
ldr rIBASE, [rSELF, #offThread_curHandlerTable] @ set rIBASE

#if defined(WITH_JIT)
.LentryInstr:
/* Entry is always a possible trace start */
ldr r0, [rSELF, #offThread_pJitProfTable]
FETCH_INST()
mov r1, #0 @ prepare the value for the new state
str r1, [rSELF, #offThread_inJitCodeCache] @ back to the interp land
cmp r0, #0 @ is profiling disabled?

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

#if !defined(WITH_SELF_VERIFICATION)
    bne    common_updateProfile    @ profiling is enabled
#else
    ldr    r2, [rSELF, #offThread_shadowSpace] @ to find out the jit exit state
    beq    1f                      @ profiling is disabled
    ldr    r3, [r2, #offShadowSpace_jitExitState] @ jit exit state
    cmp    r3, #kSVSTraceSelect    @ hot trace following?
    moveq   r2, #kJitTSelectRequestHot @ ask for trace selection
    beq    common_selectTrace      @ go build the trace
    cmp    r3, #kSVSNoProfile      @ don't profile the next instruction?
    beq    1f                      @ intepret the next instruction
    b      common_updateProfile    @ collect profiles
#endif
1:
    GET_INST_OPCODE(ip)
    GOTO_OPCODE(ip)
#else
    /* start executing the instruction at rPC */
    FETCH_INST()                @ load rINST from rPC
    GET_INST_OPCODE(ip)         @ extract opcode from rINST
    GOTO_OPCODE(ip)             @ jump to next instruction
#endif

.Lbad_arg:
    ldr    r0, strBadEntryPoint
0: add    r0, pc
    @ r1 holds value of entryPoint
    bl    printf
    bl    dvmAbort
    .fnend
    .size  dvmMterpStdRun, .-dvmMterpStdRun

strBadEntryPoint:
    .word  PCREL_REF(.LstrBadEntryPoint,0b)

    .global dvmMterpStdBail
    .type  dvmMterpStdBail, %function

```

xxvii \platform\_dalvik-master\vm\mterp\out\InterpAsm-armv5te.S

```

/*
 * Interpreter entry point.
 */

/*
 * We don't have formal stack frames, so gdb scans upward in the code
 * to find the start of the function (a label with the %function type),
 * and then looks at the next few instructions to figure out what
 * got pushed onto the stack. From this it figures out how to restore

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* the registers, including PC, for the previous stack frame. If gdb
* sees a non-function label, it stops scanning, so either we need to
* have nothing but assembler-local labels between the entry point and
* the break, or we need to fake it out.
*
* When this is defined, we add some stuff to make gdb less confused.
*/
#define ASSIST_DEBUGGER 1

.text
.align 2
.global dvmMterpStdRun
.type dvmMterpStdRun, %function

/*
* On entry:
* r0 Thread* self
*
* The return comes via a call to dvmMterpStdBail().
*/
dvmMterpStdRun:
#define MTERP_ENTRY1 \
    .save {r4-r10,fp,lr}; \
    stmfd sp!, {r4-r10,fp,lr} @ save 9 regs
#define MTERP_ENTRY2 \
    .pad #4; \
    sub sp, sp, #4 @ align 64

    .fnstart
    MTERP_ENTRY1
    MTERP_ENTRY2

    /* save stack pointer, add magic word for debuggerd */
    str sp, [r0, #offThread_bailPtr] @ save SP for eventual return

    /* set up "named" registers, figure out entry point */
    mov rSELF, r0 @ set rSELF
    LOAD_PC_FP_FROM_SELF() @ load rPC and rFP from "thread"
    ldr rIBASE, [rSELF, #offThread_curHandlerTable] @ set rIBASE

#if defined(WITH_JIT)
.LentryInstr:
    /* Entry is always a possible trace start */
    ldr r0, [rSELF, #offThread_pJitProfTable]
    FETCH_INST()
    mov r1, #0 @ prepare the value for the new state
    str r1, [rSELF, #offThread_inJitCodeCache] @ back to the interp land
    cmp r0, #0 @ is profiling disabled?
#endif
#if !defined(WITH_SELF_VERIFICATION)
    bne common_updateProfile @ profiling is enabled

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

#else
    ldr    r2, [rSELF, #offThread_shadowSpace] @ to find out the jit exit state
    beq    1f                                @ profiling is disabled
    ldr    r3, [r2, #offShadowSpace_jitExitState] @ jit exit state
    cmp    r3, #kSVSTraceSelect @ hot trace following?
    moveq   r2, #kJitTSelectRequestHot @ ask for trace selection
    beq    common_selectTrace @ go build the trace
    cmp    r3, #kSVSNoProfile @ don't profile the next instruction?
    beq    1f                                @ intepret the next instruction
    b      common_updateProfile @ collect profiles
#endif
1:
    GET_INST_OPCODE(ip)
    GOTO_OPCODE(ip)
#else
    /* start executing the instruction at rPC */
    FETCH_INST() @ load rINST from rPC
    GET_INST_OPCODE(ip) @ extract opcode from rINST
    GOTO_OPCODE(ip) @ jump to next instruction
#endif

.Lbad_arg:
    ldr    r0, strBadEntryPoint
0: add    r0, pc
    @ r1 holds value of EntryPoint
    bl     printf
    bl     dvmAbort
.fnend
.size    dvmMterpStdRun, .-dvmMterpStdRun

strBadEntryPoint:
    .word  PCREL_REF(.LstrBadEntryPoint,0b)

.global dvmMterpStdBail
.type   dvmMterpStdBail, %function

```

xxviii \platform\_dalvik-master\vm\mterp\out\InterpAsm-armv7-a-neon.S

```

/*
 * Interpreter entry point.
 */

/*
 * We don't have formal stack frames, so gdb scans upward in the code
 * to find the start of the function (a label with the %function type),
 * and then looks at the next few instructions to figure out what
 * got pushed onto the stack. From this it figures out how to restore
 * the registers, including PC, for the previous stack frame. If gdb
 * sees a non-function label, it stops scanning, so either we need to

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* have nothing but assembler-local labels between the entry point and
* the break, or we need to fake it out.
*
* When this is defined, we add some stuff to make gdb less confused.
*/
#define ASSIST_DEBUGGER 1

.text
.align 2
.global dvmMterpStdRun
.type dvmMterpStdRun, %function

/*
* On entry:
* r0 Thread* self
*
* The return comes via a call to dvmMterpStdBail().
*/
dvmMterpStdRun:
#define MTERP_ENTRY1 \
    .save {r4-r10,fp,lr}; \
    stmfd sp!, {r4-r10,fp,lr} @ save 9 regs
#define MTERP_ENTRY2 \
    .pad #4; \
    sub sp, sp, #4 @ align 64

.fnstart
MTERP_ENTRY1
MTERP_ENTRY2

/* save stack pointer, add magic word for debuggerd */
str sp, [r0, #offThread_bailPtr] @ save SP for eventual return

/* set up "named" registers, figure out entry point */
mov rSELF, r0 @ set rSELF
LOAD_PC_FP_FROM_SELF() @ load rPC and rFP from "thread"
ldr rIBASE, [rSELF, #offThread_curHandlerTable] @ set rIBASE

#if defined(WITH_JIT)
.LentryInstr:
/* Entry is always a possible trace start */
ldr r0, [rSELF, #offThread_pJitProfTable]
FETCH_INST()
mov r1, #0 @ prepare the value for the new state
str r1, [rSELF, #offThread_inJitCodeCache] @ back to the interp land
cmp r0, #0 @ is profiling disabled?
#if !defined(WITH_SELF_VERIFICATION)
bne common_updateProfile @ profiling is enabled
#else
ldr r2, [rSELF, #offThread_shadowSpace] @ to find out the jit exit state

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

```

    beq    1f                @ profiling is disabled
    ldr    r3, [r2, #offShadowSpace_jitExitState] @ jit exit state
    cmp    r3, #kSVSTraceSelect @ hot trace following?
    moveq  r2, #kJiTSelectRequestHot @ ask for trace selection
    beq    common_selectTrace @ go build the trace
    cmp    r3, #kSVSNoProfile @ don't profile the next instruction?
    beq    1f                @ intepret the next instruction
    b      common_updateProfile @ collect profiles
#endif
1:
    GET_INST_OPCODE(ip)
    GOTO_OPCODE(ip)
#else
    /* start executing the instruction at rPC */
    FETCH_INST() @ load rINST from rPC
    GET_INST_OPCODE(ip) @ extract opcode from rINST
    GOTO_OPCODE(ip) @ jump to next instruction
#endif

.Lbad_arg:
    ldr    r0, strBadEntryPoint
0: add    r0, pc
    @ r1 holds value of EntryPoint
    bl     printf
    bl     dvmAbort
    .fnend
    .size  dvmMterpStdRun, .-dvmMterpStdRun

strBadEntryPoint:
    .word  PCREL_REF(.LstrBadEntryPoint,0b)

    .global dvmMterpStdBail
    .type  dvmMterpStdBail, %function

```

xxix \platform\_dalvik-master\vm\mterp\out\InterpAsm-armv7-a.S

```

/*
 * Interpreter entry point.
 */

/*
 * We don't have formal stack frames, so gdb scans upward in the code
 * to find the start of the function (a label with the %function type),
 * and then looks at the next few instructions to figure out what
 * got pushed onto the stack. From this it figures out how to restore
 * the registers, including PC, for the previous stack frame. If gdb
 * sees a non-function label, it stops scanning, so either we need to
 * have nothing but assembler-local labels between the entry point and
 * the break, or we need to fake it out.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

*
* When this is defined, we add some stuff to make gdb less confused.
*/
#define ASSIST_DEBUGGER 1

.text
.align 2
.global dvmMterpStdRun
.type dvmMterpStdRun, %function

/*
* On entry:
* r0 Thread* self
*
* The return comes via a call to dvmMterpStdBail().
*/
dvmMterpStdRun:
#define MTERP_ENTRY1 \
    .save {r4-r10,fp,lr}; \
    stmfd sp!, {r4-r10,fp,lr} @ save 9 regs
#define MTERP_ENTRY2 \
    .pad #4; \
    sub sp, sp, #4 @ align 64

    .fnstart
    MTERP_ENTRY1
    MTERP_ENTRY2

/* save stack pointer, add magic word for debuggerd */
str sp, [r0, #offThread_bailPtr] @ save SP for eventual return

/* set up "named" registers, figure out entry point */
mov rSELF, r0 @ set rSELF
LOAD_PC_FP_FROM_SELF() @ load rPC and rFP from "thread"
ldr rIBASE, [rSELF, #offThread_curHandlerTable] @ set rIBASE

#if defined(WITH_JIT)
.LentryInstr:
/* Entry is always a possible trace start */
ldr r0, [rSELF, #offThread_pJitProfTable]
FETCH_INST()
mov r1, #0 @ prepare the value for the new state
str r1, [rSELF, #offThread_inJitCodeCache] @ back to the interp land
cmp r0, #0 @ is profiling disabled?
#if !defined(WITH_SELF_VERIFICATION)
    bne common_updateProfile @ profiling is enabled
#else
    ldr r2, [rSELF, #offThread_shadowSpace] @ to find out the jit exit state
    beq 1f @ profiling is disabled
    ldr r3, [r2, #offShadowSpace_jitExitState] @ jit exit state

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    cmp    r3, #kSVSTraceSelect    @ hot trace following?
    moveq  r2, #kJiTSelectRequestHot @ ask for trace selection
    beq    common_selectTrace      @ go build the trace
    cmp    r3, #kSVSNoProfile      @ don't profile the next instruction?
    beq    1f                      @ intepret the next instruction
    b      common_updateProfile     @ collect profiles
#endif
1:
    GET_INST_OPCODE(ip)
    GOTO_OPCODE(ip)
#else
    /* start executing the instruction at rPC */
    FETCH_INST()                  @ load rINST from rPC
    GET_INST_OPCODE(ip)           @ extract opcode from rINST
    GOTO_OPCODE(ip)               @ jump to next instruction
#endif

.Lbad_arg:
    ldr    r0, strBadEntryPoint
0: add    r0, pc
    @ r1 holds value of EntryPoint
    bl    printf
    bl    dvmAbort
.fend
.size    dvmMterpStdRun, .-dvmMterpStdRun

strBadEntryPoint:
    .word  PCREL_REF(.LstrBadEntryPoint, 0b)

.global dvmMterpStdBail
.type   dvmMterpStdBail, %function

xxx \platform_dalvik-master\vm\mterp\out\InterpAsm-mips.S

/*
 * Interpreter entry point.
 */

#define ASSIST_DEBUGGER 1

    .text
    .align 2
    .global dvmMterpStdRun
    .ent dvmMterpStdRun
    .frame sp, STACK_SIZE, ra
/*
 * On entry:
 * r0 Thread* self
 *

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* The return comes via a call to dvmMterpStdBail().
*/

dvmMterpStdRun:
    .set noreorder
    .cpload t9
    .set reorder
/* Save to the stack. Frame size = STACK_SIZE */
    STACK_STORE_FULL()
/* This directive will make sure all subsequent jal restore gp at a known offset */
    .cprestore STACK_OFFSET_GP

    addu    fp, sp, STACK_SIZE      # Move Frame Pointer to the base of frame
/* save stack pointer, add magic word for debuggerd */
    sw      sp, offThread_bailPtr(a0)  # Save SP

/* set up "named" registers, figure out entry point */
    move    rSELF, a0                # set rSELF
    LOAD_PC_FROM_SELF()
    LOAD_FP_FROM_SELF()
    lw      rIBASE, offThread_curHandlerTable(rSELF)

#if defined(WITH_JIT)
.LentryInstr:
/* Entry is always a possible trace start */
    lw      a0, offThread_pJitProfTable(rSELF)
    FETCH_INST()                     # load rINST from rPC
    sw      zero, offThread_inJitCodeCache(rSELF)
#if !defined(WITH_SELF_VERIFICATION)
    bnez    a0, common_updateProfile  # profiling is enabled
#else
    lw      a2, offThread_shadowSpace(rSELF) # to find out the jit exit state
    beqz    a0, 1f                     # profiling is disabled
    lw      a3, offShadowSpace_jitExitState(a2) # jit exit state
    li      t0, kSVSTraceSelect
    bne     a3, t0, 2f
    li      a2, kJitTSelectRequestHot  # ask for trace selection
    b       common_selectTrace         # go build the trace
2:
    li      a4, kSVSNoProfile
    beq     a3, a4, 1f                 # don't profile the next instruction?
    b       common_updateProfile       # collect profiles
#endif
1:
    GET_INST_OPCODE(t0)                # extract opcode from rINST
    GOTO_OPCODE(t0)                   # jump to next instruction
#else
/* start executing the instruction at rPC */
    FETCH_INST()                      # load rINST from rPC
    GET_INST_OPCODE(t0)               # extract opcode from rINST

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```
GOTO_OPCODE(t0)          # jump to next instruction
#endif
```

```
.Lbad_arg:
    la      a0, .LstrBadEntryPoint
    #a1 holds value of entryPoint
    JAL(sprintf)
    JAL(dvmAbort)
```

```
.end dvmMterpStdRun
```

```
.global dvmMterpStdBail
.ent dvmMterpStdBail
```

```
xxxi \platform_dalvik-master\vm\mterp\out\InterpAsm-x86.S
```

```
/*
 * bool dvmMterpStdRun(Thread* self)
 *
 * Interpreter entry point. Returns changeInterp.
 *
 */
dvmMterpStdRun:
    push    %ebp          # save caller base pointer
    movl    %esp, %ebp    # set our %ebp
    movl    rSELF, %ecx    # get incoming rSELF
/*
 * At this point we've allocated one slot on the stack
 * via push and stack is 8-byte aligned. Allocate space
 * for 9 spill slots, 4 local slots, 5 arg slots to bring
 * us to 16-byte alignment
 */
    subl    $(FRAME_SIZE-4), %esp

/* Spill callee save regs */
    movl    %edi,EDI_SPILL(%ebp)
    movl    %esi,ESI_SPILL(%ebp)
    movl    %ebx,EBX_SPILL(%ebp)

/* Set up "named" registers */
    movl    offThread_pc(%ecx),rPC
    movl    offThread_curFrame(%ecx),rFP
    movl    offThread_curHandlerTable(%ecx),rIBASE

/* Remember %esp for future "longjmp" */
    movl    %esp,offThread_bailPtr(%ecx)

/* Fetch next instruction before potential jump */
    FETCH_INST
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

#if defined(WITH_JIT)
    GET_JIT_PROF_TABLE %ecx %eax
    movl    $0, offThread_inJitCodeCache(%ecx)
    cmpl    $0, %eax
    jne     common_updateProfile # set up %ebx & %edx & rPC
#endif

/* Normal case: start executing the instruction at rPC */
GOTO_NEXT

.global dvmMterpStdBail
.type   dvmMterpStdBail, %function

xxxii \platform_dalvik-master\vm\mterp\out\InterpC-allstubs.cpp

/*
 * C mterp entry point. This just calls the various C fallbacks, making
 * this a slow but portable interpreter.
 *
 * This is only used for the "allstubs" variant.
 */
void dvmMterpStdRun(Thread* self)
{
    jmp_buf jmpBuf;

    self->interpSave.bailPtr = &jmpBuf;

    /* We exit via a longjmp */
    if (setjmp(jmpBuf)) {
        LOGVV("mterp threadid=%d returning", dvmThreadSelf()->threadId);
        return;
    }

    /* run until somebody longjmp()s out */
    while (true) {
        typedef void (*Handler)(Thread* self);

        u2 inst = /*self->interpSave.* /pc[0];
        /*
         * In mterp, dvmCheckBefore is handled via the altHandlerTable,
         * while in the portable interpreter it is part of the handler
         * FINISH code. For allstubs, we must do an explicit check
         * in the interpretation loop.
         */
        if (self->interpBreak.ctl.subMode) {
            dvmCheckBefore(pc, fp, self);
        }
        Handler handler = (Handler) gDvmMterpHandlers[inst & 0xff];
        (void) gDvmMterpHandlerNames; /* avoid gcc "defined but not used" */
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    LOGVV("handler %p %s",
        handler, (const char*) gDvmMterpHandlerNames[inst & 0xff]);
    (*handler)(self);
}
}

```

```

xxxiii    \platform_dalvik-master\vm\mterp\out\InterpC-portable.cpp

```

```

/*--- start of opcodes ---*/

```

```

/* File: c/OP_CONST_CLASS.cpp */

```

```

HANDLE_OPCODE(OP_CONST_CLASS/*vAA, class@BBBB*/)

```

```

{
    ClassObject* clazz;

```

```

    vdst = INST_AA(inst);
    ref = FETCH(1);

```

```

    ILOGV("|const-class v%d class@0x%04x", vdst, ref);
    clazz = dvmDexGetResolvedClass(methodClassDex, ref);

```

```

    if (clazz == NULL) {
        EXPORT_PC();

```

```

    clazz = dvmResolveClass(curMethod->clazz, ref, true);

```

```

    if (clazz == NULL)
        GOTO_exceptionThrown();

```

```

    }
    SET_REGISTER(vdst, (u4) clazz);

```

```

}
    FINISH(2);
    OP_END

```

```

GOTO_TARGET(invokerStatic, boolmethodCallRange)
    EXPORT_PC();

```

```

    vsrc1 = INST_AA(inst);    /* AA (count) or BA (count + arg 5) */
    ref = FETCH(1);          /* method ref */
    vdst = FETCH(2);          /* 4 regs -or- first reg */

```

```

    if (methodCallRange)
        ILOGV("|invoke-static-range args=%d @0x%04x {regs=v%d-v%d}",
            vsrc1, ref, vdst, vdst+vsrc1-1);

```

```

    else
        ILOGV("|invoke-static args=%d @0x%04x {regs=0x%04x %x}",
            vsrc1 >> 4, ref, vdst, vsrc1 & 0x0f);

```

```

    methodToCall = dvmDexGetResolvedMethod(methodClassDex, ref);
    if (methodToCall == NULL) {
        methodToCall = dvmResolveMethod(curMethod->clazz, ref, METHOD_STATIC);
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

if (methodToCall == NULL) {
    ILOGV("+ unknown method");
    GOTO_exceptionThrown();
}

```

xxxiv platform\_dalvik-master\vm\DvmDex.cpp

```

/*
 * Create auxillary data structures.
 *
 * We need a 4-byte pointer for every reference to a class, method, field,
 * or string constant. Summed up over all loaded DEX files (including the
 * whoppers in the bootstrap class path), this adds up to be quite a bit
 * of native memory.
 *
 * For more traditional VMs these values could be stuffed into the loaded
 * class file constant pool area, but we don't have that luxury since our
 * classes are memory-mapped read-only.
 *
 * The DEX optimizer will remove the need for some of these (e.g. we won't
 * use the entry for virtual methods that are only called through
 * invoke-virtual-quick), creating the possibility of some space reduction
 * at dexopt time.
 */

```

```

static DvmDex* allocateAuxStructures(DexFile* pDexFile)
{
    DvmDex* pDvmDex;
    const DexHeader* pHeader;
    u4 stringSize, classSize, methodSize, fieldSize;

    pHeader = pDexFile->pHeader;

    stringSize = pHeader->stringIdsSize * sizeof(structStringObject*);
    classSize = pHeader->typeIdsSize * sizeof(structClassObject*);
    methodSize = pHeader->methodIdsSize * sizeof(structMethod*);
    fieldSize = pHeader->fieldIdsSize * sizeof(structField*);

    u4 totalSize = sizeof(DvmDex) +
        stringSize + classSize + methodSize + fieldSize;

    u1 *blob = (u1 *)dvmAllocRegion(totalSize,
        PROT_READ | PROT_WRITE, "dalvik-aux-structure");
    if ((void *)blob == MAP_FAILED)
        return NULL;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

pDvmDex = (DvmDex*)blob;
blob += sizeof(DvmDex);

pDvmDex->pDexFile = pDexFile;
pDvmDex->pHeader = pHeader;

pDvmDex->pResStrings = (structStringObject**)blob;
blob += stringSize;
pDvmDex->pResClasses = (structClassObject**)blob;
blob += classSize;
pDvmDex->pResMethods = (structMethod**)blob;
blob += methodSize;
pDvmDex->pResFields = (structField**)blob;

ALOGV("+++ DEX %p: allocateAux (%d+%d+%d+%d)*4 = %d bytes",
      pDvmDex, stringSize/4, classSize/4, methodSize/4, fieldSize/4,
      stringSize + classSize + methodSize + fieldSize);

pDvmDex->pInterfaceCache = dvmAllocAtomicCache(DEX_INTERFACE_CACHE_SIZE);

dvmInitMutex(&pDvmDex->modLock);

return pDvmDex;
}

/*
 * Given an open optimized DEX file, map it into read-only shared memory and
 * parse the contents.
 *
 * Returns nonzero on error.
 */
intdvmDexFileOpenFromFd(int fd, DvmDex** ppDvmDex)
{
    DvmDex* pDvmDex;
    DexFile* pDexFile;
    MemMapping memMap;
    int parseFlags = kDexParseDefault;
    int result = -1;

    if (gDvm.verifyDexChecksum)
        parseFlags |= kDexParseVerifyChecksum;

    if (lseek(fd, 0, SEEK_SET) < 0) {
        ALOGE("lseek rewind failed");
        goto bail;
    }

    if (sysMapFileInShmemWritableReadOnly(fd, &memMap) != 0) {
        ALOGE("Unable to map file");
        goto bail;
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    pDexFile = dexFileParse((u1*)memMap.addr, memMap.length, parseFlags);
    if (pDexFile == NULL) {
        ALOGE("DEX parse failed");
        sysReleaseShmem(&memMap);
    goto bail;
    }

    pDvmDex = allocateAuxStructures(pDexFile);
    if (pDvmDex == NULL) {
        dexFileFree(pDexFile);
        sysReleaseShmem(&memMap);
    goto bail;
    }

    /* tuck this into the DexFile so it gets released later */
    sysCopyMap(&pDvmDex->memMap, &memMap);
    pDvmDex->isMappedReadOnly = true;
    *ppDvmDex = pDvmDex;
    result = 0;

bail:
    return result;
}

```

```

xxxv      platform_dalvik-master\dexopt\OptMain.cpp

```

```

/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* Command-line DEX optimization and verification entry point.
*
* There are three ways to launch this:
* (1) From the VM. This takes a dozen args, one of which is a file
*     descriptor that acts as both input and output. This allows us to
*     remain ignorant of where the DEX data originally came from.
* (2) From installD or another native application. Pass in a file
*     descriptor for a zip file, a file descriptor for the output, and
*     a filename for debug messages. Many assumptions are made about
*     what's going on (verification + optimization are enabled, boot
*     class path is in BOOTCLASSPATH, etc).
* (3) On the host during a build for preoptimization. This behaves
*     almost the same as (2), except it takes file names instead of
*     file descriptors.
*
* There are some fragile aspects around bootclasspath entries, owing
* largely to the VM's history of working on whenever it thought it needed
* instead of strictly doing what it was told. If optimizing bootclasspath
* entries, always do them in the order in which they appear in the path.
*/
#include "Dalvik.h"
#include "libdex/OptInvocation.h"

#include "cutils/log.h"
#include "cutils/process_name.h"

#include <fcntl.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

static const char* kClassesDex = "classes.dex";

/*
* Extract "classes.dex" from zipFd into "cacheFd", leaving a little space
* up front for the DEX optimization header.
*/
static int extractAndProcessZip(int zipFd, int cacheFd,
    const char* debugFileName, bool isBootstrap, const char* bootClassPath,
    const char* dexoptFlagStr)
{
    ZipArchive zippy;
    ZipEntry zipEntry;
    size_t uncompLen;
    long modWhen, crc32;
    off_t dexOffset;
    int err;
    int result = -1;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

int dexoptFlags = 0;      /* bit flags, from enum DexoptFlags */
DexClassVerifyMode verifyMode = VERIFY_MODE_ALL;
DexOptimizerMode dexOptMode = OPTIMIZE_MODE_VERIFIED;

memset(&zippy, 0, sizeof(zippy));

/* make sure we're still at the start of an empty file */
if (lseek(cacheFd, 0, SEEK_END) != 0) {
    ALOGE("DexOptZ: new cache file '%s' is not empty", debugFileName);
    goto bail;
}

/*
 * Write a skeletal DEX optimization header. We want the classes.dex
 * to come just after it.
 */
err = dexOptCreateEmptyHeader(cacheFd);
if (err != 0)
    goto bail;

/* record the file position so we can get back here later */
dexOffset = lseek(cacheFd, 0, SEEK_CUR);
if (dexOffset < 0)
    goto bail;

/*
 * Open the zip archive, find the DEX entry.
 */
if (dexZipPrepArchive(zipFd, debugFileName, &zippy) != 0) {
    ALOGW("DexOptZ: unable to open zip archive '%s'", debugFileName);
    goto bail;
}

zipEntry = dexZipFindEntry(&zippy, kClassesDex);
if (zipEntry == NULL) {
    ALOGW("DexOptZ: zip archive '%s' does not include %s",
        debugFileName, kClassesDex);
    goto bail;
}

/*
 * Extract some info about the zip entry.
 */
if (dexZipGetEntryInfo(&zippy, zipEntry, NULL, &uncompLen, NULL, NULL,
    &modWhen, &crc32) != 0)
{
    ALOGW("DexOptZ: zip archive GetEntryInfo failed on %s", debugFileName);
    goto bail;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

uncompLen = uncompLen;
modWhen = modWhen;
crc32 = crc32;

/*
 * Extract the DEX data into the cache file at the current offset.
 */
if (dexZipExtractEntryToFile(&zippy, zipEntry, cacheFd) != 0) {
    ALOGW("DexOptZ: extraction of %s from %s failed",
        kClassesDex, debugFileName);
    goto bail;
}

/* Parse the options. */
if (dexoptFlagStr[0] != '\0') {
    const char* opc;
    const char* val;

    opc = strstr(dexoptFlagStr, "v=");    /* verification */
    if (opc != NULL) {
        switch (*(opc+2)) {
            case 'n': verifyMode = VERIFY_MODE_NONE;        break;
            case 'r': verifyMode = VERIFY_MODE_REMOTE;      break;
            case 'a': verifyMode = VERIFY_MODE_ALL;         break;
            default:  break;
        }
    }

    opc = strstr(dexoptFlagStr, "o=");    /* optimization */
    if (opc != NULL) {
        switch (*(opc+2)) {
            case 'n': dexOptMode = OPTIMIZE_MODE_NONE;      break;
            case 'v': dexOptMode = OPTIMIZE_MODE_VERIFIED;  break;
            case 'a': dexOptMode = OPTIMIZE_MODE_ALL;       break;
            case 'f': dexOptMode = OPTIMIZE_MODE_FULL;      break;
            default:  break;
        }
    }

    opc = strstr(dexoptFlagStr, "m=y");    /* register map */
    if (opc != NULL) {
        dexoptFlags |= DEXOPT_GEN_REGISTER_MAPS;
    }

    opc = strstr(dexoptFlagStr, "u=");    /* uniprocessor target */
    if (opc != NULL) {
        switch (*(opc+2)) {
            case 'y': dexoptFlags |= DEXOPT_UNIPROCESSOR;  break;
            case 'n': dexoptFlags |= DEXOPT_SMP;           break;
            default:  break;
        }
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }
  }
}

/*
 * Prep the VM and perform the optimization.
 */

if (dvmPrepForDexOpt(bootClassPath, dexOptMode, verifyMode,
    dexoptFlags) != 0)
{
    ALOGE("DexOptZ: VM init failed");
    goto bail;
}

//vmStarted = 1;

/* do the optimization */
if (!dvmContinueOptimization(cacheFd, dexOffset, uncompLen, debugFileName,
    modWhen, crc32, isBootstrap))
{
    ALOGE("Optimization failed");
    goto bail;
}

/* we don't shut the VM down -- process is about to exit */

result = 0;

bail:
    dexZipCloseArchive(&zippy);
    return result;
}

/*
 * Common functionality for normal device-side processing as well as
 * preoptimization.
 */
static int processZipFile(int zipFd, int cacheFd, const char* zipName,
    const char *dexoptFlags)
{
    char* bcpCopy = NULL;

    /*
     * Check to see if this is a bootstrap class entry. If so, truncate
     * the path.
     */
    const char* bcp = getenv("BOOTCLASSPATH");
    if (bcp == NULL) {
        ALOGE("DexOptZ: BOOTCLASSPATH not set");
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    return -1;
}

bool isBootstrap = false;
const char* match = strstr(bcp, zipName);
if (match != NULL) {
    /*
     * TODO: we have a partial string match, but that doesn't mean
     * we've matched an entire path component. We should make sure
     * that we're matching on the full zipName, and if not we
     * should re-do the strstr starting at (match+1).
     *
     * The scenario would be a bootclasspath with something like
     * "/system/framework/core.jar" while we're trying to optimize
     * "/framework/core.jar". Not very likely since all paths are
     * absolute and end with ".jar", but not impossible.
     */
    int matchOffset = match - bcp;
    if (matchOffset > 0 && bcp[matchOffset-1] == ':')
        matchOffset--;
    ALOGV("DexOptZ: found '%s' in bootclasspath, cutting off at %d",
        zipName, matchOffset);
    bcpCopy = strdup(bcp);
    bcpCopy[matchOffset] = '\0';

    bcp = bcpCopy;
    ALOGD("DexOptZ: truncated BOOTCLASSPATH to '%s'", bcp);
    isBootstrap = true;
}

int result = extractAndProcessZip(zipFd, cacheFd, zipName, isBootstrap,
    bcp, dexoptFlags);

free(bcpCopy);
return result;
}

/* advance to the next arg and extract it */
#define GET_ARG(_var, _func, _msg) \
{ \
    char* endp; \
    (_var) = _func(++argv, &endp, 0); \
    if (*endp != '\0') { \
        ALOGE("%s '%s'", _msg, *argv); \
        goto bail; \
    } \
    --argc; \
}

/*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* Parse arguments. We want:
* 0. (name of dexopt command -- ignored)
* 1. "--zip"
* 2. zip fd (input, read-only)
* 3. cache fd (output, read-write, locked with flock)
* 4. filename of zipfile being optimized (used for debug messages and
*    for comparing against BOOTCLASSPATH; does not need to be
*    accessible or even exist)
* 5. dexopt flags
*
* The BOOTCLASSPATH environment variable is assumed to hold the correct
* boot class path. If the filename provided appears in the boot class
* path, the path will be truncated just before that entry (so that, if
* you were to dexopt "core.jar", your bootclasspath would be empty).
*
* This does not try to normalize the boot class path name, so the
* filename test won't catch you if you get creative.
*/
static int fromZip(int argc, char* const argv[])
{
    int result = -1;
    int zipFd, cacheFd;
    const char* zipName;
    char* bcpCopy = NULL;
    const char* dexoptFlags;

    if (argc != 6) {
        ALOGE("Wrong number of args for --zip (found %d)", argc);
        goto bail;
    }

    /* skip "--zip" */
    argc--;
    argv++;

    GET_ARG(zipFd, strtol, "bad zip fd");
    GET_ARG(cacheFd, strtol, "bad cache fd");
    zipName = *++argv;
    --argc;
    dexoptFlags = *++argv;
    --argc;

    result = processZipFile(zipFd, cacheFd, zipName, dexoptFlags);

bail:
    return result;
}

/*
* Parse arguments for a preoptimization run. This is when dalvikvm is run

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* on a host to optimize dex files for eventual running on a (different)
* device. We want:
* 0. (name of dexopt command -- ignored)
* 1. "--preopt"
* 2. zipfile name
* 3. output file name
* 4. dexopt flags
*
* The BOOTCLASSPATH environment variable is assumed to hold the correct
* boot class path. If the filename provided appears in the boot class
* path, the path will be truncated just before that entry (so that, if
* you were to dexopt "core.jar", your bootclasspath would be empty).
*
* This does not try to normalize the boot class path name, so the
* filename test won't catch you if you get creative.
*/
static int preopt(int argc, char* const argv[])
{
    int zipFd = -1;
    int outFd = -1;
    int result = -1;

    if (argc != 5) {
        /*
         * Use stderr here, since this variant is meant to be called on
         * the host side.
         */
        fprintf(stderr, "Wrong number of args for --preopt (found %d)\n",
            argc);
        return -1;
    }

    const char* zipName = argv[2];
    const char* outName = argv[3];
    const char* dexoptFlags = argv[4];

    if (strstr(dexoptFlags, "u=y") == NULL &&
        strstr(dexoptFlags, "u=n") == NULL)
    {
        fprintf(stderr, "Either 'u=y' or 'u=n' must be specified\n");
        return -1;
    }

    zipFd = open(zipName, O_RDONLY);
    if (zipFd < 0) {
        perror(argv[0]);
        return -1;
    }

    outFd = open(outName, O_RDWR | O_EXCL | O_CREAT, 0666);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    if (outFd < 0) {
        perror(argv[0]);
        goto bail;
    }

    result = processZipFile(zipFd, outFd, zipName, dexoptFlags);

bail:
    if (zipFd >= 0) {
        close(zipFd);
    }

    if (outFd >= 0) {
        close(outFd);
    }

    return result;
}

/*
 * Parse arguments for an "old-style" invocation directly from the VM.
 *
 * Here's what we want:
 * 0. (name of dexopt command -- ignored)
 * 1. "--dex"
 * 2. DALVIK_VM_BUILD value, as a sanity check
 * 3. file descriptor, locked with flock, for DEX file being optimized
 * 4. DEX offset within file
 * 5. DEX length
 * 6. filename of file being optimized (for debug messages only)
 * 7. modification date of source (goes into dependency section)
 * 8. CRC of source (goes into dependency section)
 * 9. flags (optimization level, isBootstrap)
 * 10. bootclasspath entry #1
 * 11. bootclasspath entry #2
 * ...
 *
 * dvmOptimizeDexFile() in dalvik/vm/analysis/DexOptimize.c builds the
 * argument list and calls this executable.
 *
 * The bootclasspath entries become the dependencies for this DEX file.
 *
 * The open file descriptor MUST NOT be for one of the bootclasspath files.
 * The parent has the descriptor locked, and we'll try to lock it again as
 * part of processing the bootclasspath. (We can catch this and return
 * an error by comparing filenames or by opening the bootclasspath files
 * and stat()ing them for inode numbers).
 */
static int fromDex(int argc, char* const argv[])
{

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

int result = -1;
bool vmStarted = false;
char* bootClassPath = NULL;
int fd, flags, vmBuildVersion;
long offset, length;
const char* debugFileName;
u4 crc, modWhen;
char* endp;
bool onlyOptVerifiedDex = false;
DexClassVerifyMode verifyMode;
DexOptimizerMode dexOptMode;

if (argc < 10) {
    /* don't have all mandatory args */
    ALOGE("Not enough arguments for --dex (found %d)", argc);
    goto bail;
}

/* skip "--dex" */
argc--;
argv++;

/*
 * Extract the args.
 */
GET_ARG(vmBuildVersion, strtol, "bad vm build");
if (vmBuildVersion != DALVIK_VM_BUILD) {
    ALOGE("DexOpt: build rev does not match VM: %d vs %d",
        vmBuildVersion, DALVIK_VM_BUILD);
    goto bail;
}
GET_ARG(fd, strtol, "bad fd");
GET_ARG(offset, strtol, "bad offset");
GET_ARG(length, strtol, "bad length");
debugFileName = *++argv;
--argc;
GET_ARG(modWhen, strtoul, "bad modWhen");
GET_ARG(crc, strtoul, "bad crc");
GET_ARG(flags, strtol, "bad flags");

ALOGV("Args: fd=%d off=%ld len=%ld name='%s' mod=%#x crc=%#x flg=%d (argc=%d)",
    fd, offset, length, debugFileName, modWhen, crc, flags, argc);
assert(argc > 0);

if (--argc == 0) {
    bootClassPath = strdup("");
} else {
    int i, bcpLen;
    char* const* argp;
    char* cp;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

bcpLen = 0;
for (i = 0, argp = argv; i < argc; i++) {
    ++argp;
    ALOGV("DEP: '%s'", *argp);
    bcpLen += strlen(*argp) + 1;
}

cp = bootClassPath = (char*) malloc(bcpLen + 1);
for (i = 0, argp = argv; i < argc; i++) {
    int strLen;

    ++argp;
    strLen = strlen(*argp);
    if (i != 0)
        *cp++ = ':';
    memcpy(cp, *argp, strLen);
    cp += strLen;
}
*cp = '\0';

assert((int) strlen(bootClassPath) == bcpLen-1);
}
ALOGV(" bootclasspath is '%s'", bootClassPath);

/* start the VM partway */

/* ugh -- upgrade these to a bit field if they get any more complex */
if ((flags & DEXOPT_VERIFY_ENABLED) != 0) {
    if ((flags & DEXOPT_VERIFY_ALL) != 0)
        verifyMode = VERIFY_MODE_ALL;
    else
        verifyMode = VERIFY_MODE_REMOTE;
} else {
    verifyMode = VERIFY_MODE_NONE;
}
if ((flags & DEXOPT_OPT_ENABLED) != 0) {
    if ((flags & DEXOPT_OPT_ALL) != 0)
        dexOptMode = OPTIMIZE_MODE_ALL;
    else
        dexOptMode = OPTIMIZE_MODE_VERIFIED;
} else {
    dexOptMode = OPTIMIZE_MODE_NONE;
}

if (dvmPrepForDexOpt(bootClassPath, dexOptMode, verifyMode, flags) != 0) {
    ALOGE("VM init failed");
    goto bail;
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

vmStarted = true;

/* do the optimization */
if (!dvmContinueOptimization(fd, offset, length, debugFileName,
    modWhen, crc, (flags & DEXOPT_IS_BOOTSTRAP) != 0))
{
    ALOGE("Optimization failed");
    goto bail;
}

result = 0;

bail:
/*
 * In theory we should gracefully shut the VM down at this point. In
 * practice that only matters if we're checking for memory leaks with
 * valgrind -- simply exiting is much faster.
 *
 * As it turns out, the DEX optimizer plays a little fast and loose
 * with class loading. We load all of the classes from a partially-
 * formed DEX file, which is unmapped when we're done. If we want to
 * do clean shutdown here, perhaps for testing with valgrind, we need
 * to skip the munmap call there.
 */
#ifdef 0
    if (vmStarted) {
        ALOGI("DexOpt shutting down, result=%d", result);
        dvmShutdown();
    }
#endif

    free(bootClassPath);
    ALOGV("DexOpt command complete (result=%d)", result);
    return result;
}

/*
 * Main entry point. Decide where to go.
 */
int main(int argc, char* const argv[])
{
    set_process_name("dexopt");

    setvbuf(stdout, NULL, _IONBF, 0);

    if (argc > 1) {
        if (strcmp(argv[1], "--zip") == 0)
            return fromZip(argc, argv);
        else if (strcmp(argv[1], "--dex") == 0)
            return fromDex(argc, argv);
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    else if (strcmp(argv[1], "--preopt") == 0)
        return preopt(argc, argv);
}

fprintf(stderr,
    "Usage:\n\n"
    "Short version: Don't use this.\n\n"
    "Slightly longer version: This system-internal tool is used to\n"
    "produce optimized dex files. See the source code for details.\n");

return 1;
}

```

xxxvi platform\_dalvik-master\dexopt\OptMain.cpp

```

/*
 * Command-line DEX optimization and verification entry point.
 *
 * There are three ways to launch this:
 * (1) From the VM. This takes a dozen args, one of which is a file
 *     descriptor that acts as both input and output. This allows us to
 *     remain ignorant of where the DEX data originally came from.
 * (2) From installD or another native application. Pass in a file
 *     descriptor for a zip file, a file descriptor for the output, and
 *     a filename for debug messages. Many assumptions are made about
 *     what's going on (verification + optimization are enabled, boot
 *     class path is in BOOTCLASSPATH, etc).
 * (3) On the host during a build for preoptimization. This behaves
 *     almost the same as (2), except it takes file names instead of
 *     file descriptors.
 *
 * There are some fragile aspects around bootclasspath entries, owing
 * largely to the VM's history of working on whenever it thought it needed
 * instead of strictly doing what it was told. If optimizing bootclasspath
 * entries, always do them in the order in which they appear in the path.
 */
#include "Dalvik.h"
#include "libdex/OptInvocation.h"

#include "cutils/log.h"
#include "cutils/process_name.h"

#include <fcntl.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

#include <string.h>

static const char* kClassesDex = "classes.dex";

/*
 * Extract "classes.dex" from zipFd into "cacheFd", leaving a little space
 * up front for the DEX optimization header.
 */
static int extractAndProcessZip(int zipFd, int cacheFd,
    const char* debugFileName, bool isBootstrap, const char* bootClassPath,
    const char* dexoptFlagStr)
{
    ZipArchive zippy;
    ZipEntry zipEntry;
    size_t uncompLen;
    long modWhen, crc32;
    off_t dexOffset;
    int err;
    int result = -1;
    int dexoptFlags = 0;    /* bit flags, from enum DexoptFlags */
    DexClassVerifyMode verifyMode = VERIFY_MODE_ALL;
    DexOptimizerMode dexOptMode = OPTIMIZE_MODE_VERIFIED;

    memset(&zippy, 0, sizeof(zippy));

    /* make sure we're still at the start of an empty file */
    if (lseek(cacheFd, 0, SEEK_END) != 0) {
        ALOGE("DexOptZ: new cache file '%s' is not empty", debugFileName);
        goto bail;
    }

    /*
     * Write a skeletal DEX optimization header. We want the classes.dex
     * to come just after it.
     */
    err = dexOptCreateEmptyHeader(cacheFd);
    if (err != 0)
        goto bail;

    /* record the file position so we can get back here later */
    dexOffset = lseek(cacheFd, 0, SEEK_CUR);
    if (dexOffset < 0)
        goto bail;

    /*
     * Open the zip archive, find the DEX entry.
     */
    if (dexZipPrepArchive(zipFd, debugFileName, &zippy) != 0) {
        ALOGW("DexOptZ: unable to open zip archive '%s'", debugFileName);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    goto bail;
}

zipEntry = dexZipFindEntry(&zipppy, kClassesDex);
if (zipEntry == NULL) {
    ALOGW("DexOptZ: zip archive '%s' does not include %s",
        debugFileName, kClassesDex);
    goto bail;
}

/*
 * Extract some info about the zip entry.
 */
if (dexZipGetEntryInfo(&zipppy, zipEntry, NULL, &uncompLen, NULL, NULL,
    &modWhen, &crc32) != 0)
{
    ALOGW("DexOptZ: zip archive GetEntryInfo failed on %s", debugFileName);
    goto bail;
}

uncompLen = uncompLen;
modWhen = modWhen;
crc32 = crc32;

/*
 * Extract the DEX data into the cache file at the current offset.
 */
if (dexZipExtractEntryToFile(&zipppy, zipEntry, cacheFd) != 0) {
    ALOGW("DexOptZ: extraction of %s from %s failed",
        kClassesDex, debugFileName);
    goto bail;
}

/* Parse the options. */
if (dexoptFlagStr[0] != '\0') {
    const char* opc;
    const char* val;

    opc = strstr(dexoptFlagStr, "v=");    /* verification */
    if (opc != NULL) {
        switch (*(opc+2)) {
            case 'n': verifyMode = VERIFY_MODE_NONE;        break;
            case 'r': verifyMode = VERIFY_MODE_REMOTE;      break;
            case 'a': verifyMode = VERIFY_MODE_ALL;         break;
            default:  break;
        }
    }

    opc = strstr(dexoptFlagStr, "o=");    /* optimization */
    if (opc != NULL) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        switch (*(opc+2)) {
        case 'n': dexOptMode = OPTIMIZE_MODE_NONE;      break;
        case 'v': dexOptMode = OPTIMIZE_MODE_VERIFIED;  break;
        case 'a': dexOptMode = OPTIMIZE_MODE_ALL;       break;
        case 'f': dexOptMode = OPTIMIZE_MODE_FULL;      break;
        default:                                     break;
        }
    }

    opc = strstr(dexoptFlagStr, "m=y"); /* register map */
    if (opc != NULL) {
        dexoptFlags |= DEXOPT_GEN_REGISTER_MAPS;
    }

    opc = strstr(dexoptFlagStr, "u="); /* uniprocessor target */
    if (opc != NULL) {
        switch (*(opc+2)) {
        case 'y': dexoptFlags |= DEXOPT_UNIPROCESSOR;  break;
        case 'n': dexoptFlags |= DEXOPT_SMP;           break;
        default:                                     break;
        }
    }
}

/*
 * Prep the VM and perform the optimization.
 */

if (dvmPrepForDexOpt(bootClassPath, dexOptMode, verifyMode,
    dexoptFlags) != 0)
{
    ALOGE("DexOptZ: VM init failed");
    goto bail;
}

//vmStarted = 1;

/* do the optimization */
if (!dvmContinueOptimization(cacheFd, dexOffset, uncompLen, debugFileName,
    modWhen, crc32, isBootstrap))
{
    ALOGE("Optimization failed");
    goto bail;
}

/* we don't shut the VM down -- process is about to exit */

result = 0;

bail:

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    dexZipCloseArchive(&zippy);
    return result;
}

/*
 * Common functionality for normal device-side processing as well as
 * preoptimization.
 */
static int processZipFile(int zipFd, int cacheFd, const char* zipName,
    const char *dexoptFlags)
{
    char* bcpCopy = NULL;

    /*
     * Check to see if this is a bootstrap class entry. If so, truncate
     * the path.
     */
    const char* bcp = getenv("BOOTCLASSPATH");
    if (bcp == NULL) {
        ALOGE("DexOptZ: BOOTCLASSPATH not set");
        return -1;
    }

    bool isBootstrap = false;
    const char* match = strstr(bcp, zipName);
    if (match != NULL) {
        /*
         * TODO: we have a partial string match, but that doesn't mean
         * we've matched an entire path component. We should make sure
         * that we're matching on the full zipName, and if not we
         * should re-do the strstr starting at (match+1).
         *
         * The scenario would be a bootclasspath with something like
         * "/system/framework/core.jar" while we're trying to optimize
         * "/framework/core.jar". Not very likely since all paths are
         * absolute and end with ".jar", but not impossible.
         */
        int matchOffset = match - bcp;
        if (matchOffset > 0 && bcp[matchOffset-1] == ':')
            matchOffset--;
        ALOGV("DexOptZ: found '%s' in bootclasspath, cutting off at %d",
            zipName, matchOffset);
        bcpCopy = strdup(bcp);
        bcpCopy[matchOffset] = '\0';

        bcp = bcpCopy;
        ALOGD("DexOptZ: truncated BOOTCLASSPATH to '%s'", bcp);
        isBootstrap = true;
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

int result = extractAndProcessZip(zipFd, cacheFd, zipName, isBootstrap,
    bcp, dexoptFlags);

free(bcpCopy);
return result;
}

/* advance to the next arg and extract it */
#define GET_ARG(_var, _func, _msg) \
{ \
    char* endp; \
    (_var) = _func(++argv, &endp, 0); \
    if (*endp != '\0') { \
        ALOGE("%s '%s'", _msg, *argv); \
        goto bail; \
    } \
    --argc; \
}

/*
 * Parse arguments. We want:
 * 0. (name of dexopt command -- ignored)
 * 1. "--zip"
 * 2. zip fd (input, read-only)
 * 3. cache fd (output, read-write, locked with flock)
 * 4. filename of zipfile being optimized (used for debug messages and
 *    for comparing against BOOTCLASSPATH; does not need to be
 *    accessible or even exist)
 * 5. dexopt flags
 *
 * The BOOTCLASSPATH environment variable is assumed to hold the correct
 * boot class path. If the filename provided appears in the boot class
 * path, the path will be truncated just before that entry (so that, if
 * you were to dexopt "core.jar", your bootclasspath would be empty).
 *
 * This does not try to normalize the boot class path name, so the
 * filename test won't catch you if you get creative.
 */
static int fromZip(int argc, char* const argv[])
{
    int result = -1;
    int zipFd, cacheFd;
    const char* zipName;
    char* bcpCopy = NULL;
    const char* dexoptFlags;

    if (argc != 6) {
        ALOGE("Wrong number of args for --zip (found %d)", argc);
        goto bail;
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/* skip "--zip" */
argc--;
argv++;

GET_ARG(zipFd, strtol, "bad zip fd");
GET_ARG(cacheFd, strtol, "bad cache fd");
zipName = *++argv;
--argc;
dexoptFlags = *++argv;
--argc;

result = processZipFile(zipFd, cacheFd, zipName, dexoptFlags);

bail:
    return result;
}

/*
 * Parse arguments for a preoptimization run. This is when dalvikvm is run
 * on a host to optimize dex files for eventual running on a (different)
 * device. We want:
 * 0. (name of dexopt command -- ignored)
 * 1. "--preopt"
 * 2. zipfile name
 * 3. output file name
 * 4. dexopt flags
 *
 * The BOOTCLASSPATH environment variable is assumed to hold the correct
 * boot class path. If the filename provided appears in the boot class
 * path, the path will be truncated just before that entry (so that, if
 * you were to dexopt "core.jar", your bootclasspath would be empty).
 *
 * This does not try to normalize the boot class path name, so the
 * filename test won't catch you if you get creative.
 */
static int preopt(int argc, char* const argv[])
{
    int zipFd = -1;
    int outFd = -1;
    int result = -1;

    if (argc != 5) {
        /*
         * Use stderr here, since this variant is meant to be called on
         * the host side.
         */
        fprintf(stderr, "Wrong number of args for --preopt (found %d)\n",
            argc);
        return -1;
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    const char* zipName = argv[2];
    const char* outName = argv[3];
    const char* dexoptFlags = argv[4];

    if (strstr(dexoptFlags, "u=y") == NULL &&
        strstr(dexoptFlags, "u=n") == NULL)
    {
        fprintf(stderr, "Either 'u=y' or 'u=n' must be specified\n");
        return -1;
    }

    zipFd = open(zipName, O_RDONLY);
    if (zipFd < 0) {
        perror(argv[0]);
        return -1;
    }

    outFd = open(outName, O_RDWR | O_EXCL | O_CREAT, 0666);
    if (outFd < 0) {
        perror(argv[0]);
        goto bail;
    }

    result = processZipFile(zipFd, outFd, zipName, dexoptFlags);

bail:
    if (zipFd >= 0) {
        close(zipFd);
    }

    if (outFd >= 0) {
        close(outFd);
    }

    return result;
}

/*
 * Parse arguments for an "old-style" invocation directly from the VM.
 *
 * Here's what we want:
 * 0. (name of dexopt command -- ignored)
 * 1. "--dex"
 * 2. DALVIK_VM_BUILD value, as a sanity check
 * 3. file descriptor, locked with flock, for DEX file being optimized
 * 4. DEX offset within file
 * 5. DEX length
 * 6. filename of file being optimized (for debug messages only)

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* 7. modification date of source (goes into dependency section)
* 8. CRC of source (goes into dependency section)
* 9. flags (optimization level, isBootstrap)
* 10. bootclasspath entry #1
* 11. bootclasspath entry #2
* ...
*
* dvmOptimizeDexFile() in dalvik/vm/analysis/DexOptimize.c builds the
* argument list and calls this executable.
*
* The bootclasspath entries become the dependencies for this DEX file.
*
* The open file descriptor MUST NOT be for one of the bootclasspath files.
* The parent has the descriptor locked, and we'll try to lock it again as
* part of processing the bootclasspath. (We can catch this and return
* an error by comparing filenames or by opening the bootclasspath files
* and stat()ing them for inode numbers).
*/

```

```

static int fromDex(int argc, char* const argv[])
{
    int result = -1;
    bool vmStarted = false;
    char* bootClassPath = NULL;
    int fd, flags, vmBuildVersion;
    long offset, length;
    const char* debugFileName;
    u4 crc, modWhen;
    char* endp;
    bool onlyOptVerifiedDex = false;
    DexClassVerifyMode verifyMode;
    DexOptimizerMode dexOptMode;

    if (argc < 10) {
        /* don't have all mandatory args */
        ALOGE("Not enough arguments for --dex (found %d)", argc);
        goto bail;
    }

    /* skip "--dex" */
    argc--;
    argv++;

    /*
     * Extract the args.
     */
    GET_ARG(vmBuildVersion, strtol, "bad vm build");
    if (vmBuildVersion != DALVIK_VM_BUILD) {
        ALOGE("DexOpt: build rev does not match VM: %d vs %d",
            vmBuildVersion, DALVIK_VM_BUILD);
        goto bail;
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

}
GET_ARG(fd, strtol, "bad fd");
GET_ARG(offset, strtol, "bad offset");
GET_ARG(length, strtol, "bad length");
debugFileName = *++argv;
--argc;
GET_ARG(modWhen, strtoul, "bad modWhen");
GET_ARG(crc, strtoul, "bad crc");
GET_ARG(flags, strtol, "bad flags");

ALOGV("Args: fd=%d off=%ld len=%ld name='%s' mod=%#x crc=%#x flg=%d (argc=%d)",
      fd, offset, length, debugFileName, modWhen, crc, flags, argc);
assert(argc > 0);

if (--argc == 0) {
    bootClassPath = strdup("");
} else {
    int i, bcpLen;
    char* const* argp;
    char* cp;

    bcpLen = 0;
    for (i = 0, argp = argv; i < argc; i++) {
        ++argp;
        ALOGV("DEP: '%s'", *argp);
        bcpLen += strlen(*argp) + 1;
    }

    cp = bootClassPath = (char*) malloc(bcpLen + 1);
    for (i = 0, argp = argv; i < argc; i++) {
        int strLen;

        ++argp;
        strLen = strlen(*argp);
        if (i != 0)
            *cp++ = ':';
        memcpy(cp, *argp, strLen);
        cp += strLen;
    }
    *cp = '\0';

    assert((int) strlen(bootClassPath) == bcpLen - 1);
}
ALOGV(" bootclasspath is '%s'", bootClassPath);

/* start the VM partway */

/* ugh -- upgrade these to a bit field if they get any more complex */
if ((flags & DEXOPT_VERIFY_ENABLED) != 0) {
    if ((flags & DEXOPT_VERIFY_ALL) != 0)

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        verifyMode = VERIFY_MODE_ALL;
    else
        verifyMode = VERIFY_MODE_REMOTE;
    } else {
        verifyMode = VERIFY_MODE_NONE;
    }
    if ((flags & DEXOPT_OPT_ENABLED) != 0) {
        if ((flags & DEXOPT_OPT_ALL) != 0)
            dexOptMode = OPTIMIZE_MODE_ALL;
        else
            dexOptMode = OPTIMIZE_MODE_VERIFIED;
    } else {
        dexOptMode = OPTIMIZE_MODE_NONE;
    }

    if (dvmPrepForDexOpt(bootClassPath, dexOptMode, verifyMode, flags) != 0) {
        ALOGE("VM init failed");
        goto bail;
    }

    vmStarted = true;

    /* do the optimization */
    if (!dvmContinueOptimization(fd, offset, length, debugFileName,
        modWhen, crc, (flags & DEXOPT_IS_BOOTSTRAP) != 0))
    {
        ALOGE("Optimization failed");
        goto bail;
    }

    result = 0;

bail:
    /*
     * In theory we should gracefully shut the VM down at this point. In
     * practice that only matters if we're checking for memory leaks with
     * valgrind -- simply exiting is much faster.
     *
     * As it turns out, the DEX optimizer plays a little fast and loose
     * with class loading. We load all of the classes from a partially-
     * formed DEX file, which is unmapped when we're done. If we want to
     * do clean shutdown here, perhaps for testing with valgrind, we need
     * to skip the munmap call there.
     */
#ifdef 0
    if (vmStarted) {
        ALOGI("DexOpt shutting down, result=%d", result);
        dvmShutdown();
    }
#endif

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    free(bootClassPath);
    ALOGV("DexOpt command complete (result=%d)", result);
    return result;
}

/*
 * Main entry point. Decide where to go.
 */
int main(int argc, char* const argv[])
{
    set_process_name("dexopt");

    setvbuf(stdout, NULL, _IONBF, 0);

    if (argc > 1) {
        if (strcmp(argv[1], "--zip") == 0)
            return fromZip(argc, argv);
        else if (strcmp(argv[1], "--dex") == 0)
            return fromDex(argc, argv);
        else if (strcmp(argv[1], "--preopt") == 0)
            return preopt(argc, argv);
    }

    fprintf(stderr,
        "Usage:\n\n"
        "Short version: Don't use this.\n\n"
        "Slightly longer version: This system-internal tool is used to\n"
        "produce optimized dex files. See the source code for details.\n");

    return 1;
}

```

xxxvii      dalvik-master\vm\analysis\DexPrepare.cpp

```

/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

```

```

/*
* Prepare a DEX file for use by the VM. Depending upon the VM options
* we will attempt to verify and/or optimize the code, possibly appending
* register maps.
*
* TODO: the format of the optimized header is currently "whatever we
* happen to write", since the VM that writes it is by definition the same
* as the VM that reads it. Still, it should be better documented and
* more rigorously structured.
*/

```

```

#include "Dalvik.h"
#include "libdex/OptInvocation.h"
#include "analysis/RegisterMap.h"
#include "analysis/Optimize.h"

```

```

#include <string>

```

```

#include <libgen.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <zlib.h>

```

```

/* fwd */
static bool rewriteDex(u1* addr, int len, bool doVerify, bool doOpt,
    DexClassLookup** ppClassLookup, DvmDex** ppDvmDex);
static bool loadAllClasses(DvmDex* pDvmDex);
static void verifyAndOptimizeClasses(DexFile* pDexFile, bool doVerify,
    bool doOpt);
static void verifyAndOptimizeClass(DexFile* pDexFile, ClassObject* clazz,
    const DexClassDef* pClassDef, bool doVerify, bool doOpt);
static void updateChecksum(u1* addr, int len, DexHeader* pHeader);
static int writeDependencies(int fd, u4 modWhen, u4 crc);
static bool writeOptData(int fd, const DexClassLookup* pClassLookup, \
    const RegisterMapBuilder* pRegMapBuilder);
static bool computeFileChecksum(int fd, off_t start, size_t length, u4* pSum);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/*
 * Get just the directory portion of the given path. Equivalent to dirname(3).
 */
static std::string saneDirName(const std::string& path) {
    size_t n = path.rfind('/');
    if (n == std::string::npos) {
        return ".";
    }
    return path.substr(0, n);
}

/*
 * Helper for dvmOpenCacheDexFile() in a known-error case: Check to
 * see if the directory part of the given path (all but the last
 * component) exists and is writable. Complain to the log if not.
 */
static bool directoryIsValid(const std::string& fileName)
{
    std::string dirName(saneDirName(fileName));

    struct stat sb;
    if (stat(dirName.c_str(), &sb) < 0) {
        ALOGE("Could not stat dex cache directory '%s': %s", dirName.c_str(), strerror(errno));
        return false;
    }

    if (!S_ISDIR(sb.st_mode)) {
        ALOGE("Dex cache directory isn't a directory: %s", dirName.c_str());
        return false;
    }

    if (access(dirName.c_str(), W_OK) < 0) {
        ALOGE("Dex cache directory isn't writable: %s", dirName.c_str());
        return false;
    }

    if (access(dirName.c_str(), R_OK) < 0) {
        ALOGE("Dex cache directory isn't readable: %s", dirName.c_str());
        return false;
    }

    return true;
}

/*
 * Return the fd of an open file in the DEX file cache area. If the cache
 * file doesn't exist or is out of date, this will remove the old entry,
 * create a new one (writing only the file header), and return with the
 * "new file" flag set.
 */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* It's possible to execute from an unoptimized DEX file directly,
* assuming the byte ordering and structure alignment is correct, but
* disadvantageous because some significant optimizations are not possible.
* It's not generally possible to do the same from an uncompressed Jar
* file entry, because we have to guarantee 32-bit alignment in the
* memory-mapped file.
*

```

```

* For a Jar/APK file (a zip archive with "classes.dex" inside), "modWhen"
* and "crc32" come from the Zip directory entry. For a stand-alone DEX
* file, it's the modification date of the file and the Adler32 from the
* DEX header (which immediately follows the magic). If these don't
* match what's stored in the opt header, we reject the file immediately.
*

```

```

* On success, the file descriptor will be positioned just past the "opt"
* file header, and will be locked with flock.  "**pCachedName" will point
* to newly-allocated storage.
*/

```

```

int dvmOpenCachedDexFile(const char* fileName, const char* cacheFileName,
    u4 modWhen, u4 crc, bool isBootstrap, bool* pNewFile, bool createlfMissing)
{

```

```

    int fd, cc;
    struct stat fdStat, fileStat;
    bool readOnly = false;

```

```

    *pNewFile = false;

```

```

retry:

```

```

    /*
    * Try to open the cache file. If we've been asked to,
    * create it if it doesn't exist.
    */

```

```

    fd = createlfMissing ? open(cacheFileName, O_CREAT|O_RDWR, 0644) : -1;

```

```

    if (fd < 0) {

```

```

        fd = open(cacheFileName, O_RDONLY, 0);

```

```

        if (fd < 0) {

```

```

            if (createlfMissing) {

```

```

                // TODO: write an equivalent of strerror_r that returns a std::string.

```

```

                const std::string errnoString(strerror(errno));

```

```

                if (directoryIsValid(cacheFileName)) {

```

```

                    ALOGE("Can't open dex cache file '%s': %s", cacheFileName, errnoString.c_str());

```

```

                }

```

```

            }

```

```

            return fd;

```

```

        }

```

```

        readOnly = true;

```

```

    } else {

```

```

        fchmod(fd, 0644);

```

```

    }

```

```

    /*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* Grab an exclusive lock on the cache file.  If somebody else is
* working on it, we'll block here until they complete.  Because
* we're waiting on an external resource, we go into VMWAIT mode.
*/
ALOGV("DexOpt: locking cache file %s (fd=%d, boot=%d)",
      cacheFileName, fd, isBootstrap);
ThreadStatus oldStatus = dvmChangeStatus(NULL, THREAD_VMWAIT);
cc = flock(fd, LOCK_EX | LOCK_NB);
if (cc != 0) {
    ALOGD("DexOpt: sleeping on flock(%s)", cacheFileName);
    cc = flock(fd, LOCK_EX);
}
dvmChangeStatus(NULL, oldStatus);
if (cc != 0) {
    ALOGE("Can't lock dex cache '%s': %d", cacheFileName, cc);
    close(fd);
    return -1;
}
ALOGV("DexOpt: locked cache file");

/*
* Check to see if the fd we opened and locked matches the file in
* the filesystem.  If they don't, then somebody else unlinked ours
* and created a new file, and we need to use that one instead.  (If
* we caught them between the unlink and the create, we'll get an
* ENOENT from the file stat.)
*/
cc = fstat(fd, &fdStat);
if (cc != 0) {
    ALOGE("Can't stat open file '%s'", cacheFileName);
    LOGVV("DexOpt: unlocking cache file %s", cacheFileName);
    goto close_fail;
}
cc = stat(cacheFileName, &fileStat);
if (cc != 0 ||
    fdStat.st_dev != fileStat.st_dev || fdStat.st_ino != fileStat.st_ino)
{
    ALOGD("DexOpt: our open cache file is stale; sleeping and retrying");
    LOGVV("DexOpt: unlocking cache file %s", cacheFileName);
    flock(fd, LOCK_UN);
    close(fd);
    usleep(250 * 1000);    /* if something is hosed, don't peg machine */
    goto retry;
}

/*
* We have the correct file open and locked.  If the file size is zero,
* then it was just created by us, and we want to fill in some fields
* in the "opt" header and set "pNewFile".  Otherwise, we want to
* verify that the fields in the header match our expectations, and

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* reset the file if they don't.
*/
if (fdStat.st_size == 0) {
    if (readOnly) {
        ALOGW("DexOpt: file has zero length and isn't writable");
        goto close_fail;
    }
    cc = dexOptCreateEmptyHeader(fd);
    if (cc != 0)
        goto close_fail;
    *pNewFile = true;
    ALOGV("DexOpt: successfully initialized new cache file");
} else {
    bool expectVerify, expectOpt;

    if (gDvm.classVerifyMode == VERIFY_MODE_NONE) {
        expectVerify = false;
    } else if (gDvm.classVerifyMode == VERIFY_MODE_REMOTE) {
        expectVerify = !isBootstrap;
    } else /*if (gDvm.classVerifyMode == VERIFY_MODE_ALL)*/ {
        expectVerify = true;
    }

    if (gDvm.dexOptMode == OPTIMIZE_MODE_NONE) {
        expectOpt = false;
    } else if (gDvm.dexOptMode == OPTIMIZE_MODE_VERIFIED ||
        gDvm.dexOptMode == OPTIMIZE_MODE_FULL) {
        expectOpt = expectVerify;
    } else /*if (gDvm.dexOptMode == OPTIMIZE_MODE_ALL)*/ {
        expectOpt = true;
    }

    ALOGV("checking deps, expecting vfy=%d opt=%d",
        expectVerify, expectOpt);

    if (!dvmCheckOptHeaderAndDependencies(fd, true, modWhen, crc,
        expectVerify, expectOpt))
    {
        if (readOnly) {
            /*
             * We could unlink and rewrite the file if we own it or
             * the "sticky" bit isn't set on the directory. However,
             * we're not able to truncate it, which spoils things. So,
             * give up now.
             */
            if (createlfMissing) {
                ALOGW("Cached DEX '%s' (%s) is stale and not writable",
                    fileName, cacheFileName);
            }
            goto close_fail;
        }
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    }

    /*
    * If we truncate the existing file before unlinking it, any
    * process that has it mapped will fail when it tries to touch
    * the pages.
    *
    * This is very important. The zygote process will have the
    * boot DEX files (core, framework, etc.) mapped early. If
    * (say) core.dex gets updated, and somebody launches an app
    * that uses App.dex, then App.dex gets reoptimized because it's
    * dependent upon the boot classes. However, dexopt will be
    * using the *new* core.dex to do the optimizations, while the
    * app will actually be running against the *old* core.dex
    * because it starts from zygote.
    *
    * Even without zygote, it's still possible for a class loader
    * to pull in an APK that was optimized against an older set
    * of DEX files. We must ensure that everything fails when a
    * boot DEX gets updated, and for general "why aren't my
    * changes doing anything" purposes its best if we just make
    * everything crash when a DEX they're using gets updated.
    */
    ALOGD("ODEX file is stale or bad; removing and retrying (%s)",
          cacheFileName);
    if (ftruncate(fd, 0) != 0) {
        ALOGW("Warning: unable to truncate cache file '%s': %s",
              cacheFileName, strerror(errno));
        /* keep going */
    }
    if (unlink(cacheFileName) != 0) {
        ALOGW("Warning: unable to remove cache file '%s': %d %s",
              cacheFileName, errno, strerror(errno));
        /* keep going; permission failure should probably be fatal */
    }
    LOGVV("DexOpt: unlocking cache file %s", cacheFileName);
    flock(fd, LOCK_UN);
    close(fd);
    goto retry;
} else {
    ALOGV("DexOpt: good deps in cache file");
}
}

assert(fd >= 0);
return fd;

close_fail:
    flock(fd, LOCK_UN);
    close(fd);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    return -1;
}

/*
 * Unlock the file descriptor.
 *
 * Returns "true" on success.
 */
bool dvmUnlockCachedDexFile(int fd)
{
    LOGVV("DexOpt: unlocking cache file fd=%d", fd);
    return (flock(fd, LOCK_UN) == 0);
}

/*
 * Given a descriptor for a file with DEX data in it, produce an
 * optimized version.
 *
 * The file pointed to by "fd" is expected to be a locked shared resource
 * (or private); we make no efforts to enforce multi-process correctness
 * here.
 *
 * "fileName" is only used for debug output. "modWhen" and "crc" are stored
 * in the dependency set.
 *
 * The "isBootstrap" flag determines how the optimizer and verifier handle
 * package-scope access checks. When optimizing, we only load the bootstrap
 * class DEX files and the target DEX, so the flag determines whether the
 * target DEX classes are given a (synthetic) non-NULL classLoader pointer.
 * This only really matters if the target DEX contains classes that claim to
 * be in the same package as bootstrap classes.
 *
 * The optimizer will need to load every class in the target DEX file.
 * This is generally undesirable, so we start a subprocess to do the
 * work and wait for it to complete.
 *
 * Returns "true" on success. All data will have been written to "fd".
 */
bool dvmOptimizeDexFile(int fd, off_t dexOffset, long dexLength,
    const char* fileName, u4 modWhen, u4 crc, bool isBootstrap)
{
    const char* lastPart = strrchr(fileName, '/');
    if (lastPart != NULL)
        lastPart++;
    else
        lastPart = fileName;

    ALOGD("DexOpt: --- BEGIN '%s' (bootstrap=%d) ---", lastPart, isBootstrap);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

pid_t pid;

/*
 * This could happen if something in our bootclasspath, which we thought
 * was all optimized, got rejected.
 */
if (gDvm.optimizing) {
    ALOGW("Rejecting recursive optimization attempt on '%s'", fileName);
    return false;
}

pid = fork();
if (pid == 0) {
    static const int kUseValgrind = 0;
    static const char* kDexOptBin = "/bin/dexopt";
    static const char* kValgrinder = "/usr/bin/valgrind";
    static const int kFixedArgCount = 10;
    static const int kValgrindArgCount = 5;
    static const int kMaxIntLen = 12; // '-' + 10dig + '\0' -OR- 0x + 8dig
    int bcpSize = dvmGetBootPathSize();
    int argc = kFixedArgCount + bcpSize
        + (kValgrindArgCount * kUseValgrind);
    const char* argv[argc+1]; // last entry is NULL
    char values[argc][kMaxIntLen];
    char* execFile;
    const char* androidRoot;
    int flags;

    /* change process groups, so we don't clash with ProcessManager */
    setpgid(0, 0);

    /* full path to optimizer */
    androidRoot = getenv("ANDROID_ROOT");
    if (androidRoot == NULL) {
        ALOGW("ANDROID_ROOT not set, defaulting to /system");
        androidRoot = "/system";
    }
    execFile = (char*)alloca(strlen(androidRoot) + strlen(kDexOptBin) + 1);
    strcpy(execFile, androidRoot);
    strcat(execFile, kDexOptBin);

    /*
     * Create arg vector.
     */
    int curArg = 0;

    if (kUseValgrind) {
        /* probably shouldn't ship the hard-coded path */
        argv[curArg++] = (char*)kValgrinder;
        argv[curArg++] = "--tool=memcheck";
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    argv[curArg++] = "--leak-check=yes";    // check for leaks too
    argv[curArg++] = "--leak-resolution=med"; // increase from 2 to 4
    argv[curArg++] = "--num-callers=16";    // default is 12
    assert(curArg == kValgrindArgCount);
}
argv[curArg++] = execFile;

argv[curArg++] = "--dex";

sprintf(values[2], "%d", DALVIK_VM_BUILD);
argv[curArg++] = values[2];

sprintf(values[3], "%d", fd);
argv[curArg++] = values[3];

sprintf(values[4], "%d", (int) dexOffset);
argv[curArg++] = values[4];

sprintf(values[5], "%d", (int) dexLength);
argv[curArg++] = values[5];

argv[curArg++] = (char*)fileName;

sprintf(values[7], "%d", (int) modWhen);
argv[curArg++] = values[7];

sprintf(values[8], "%d", (int) crc);
argv[curArg++] = values[8];

flags = 0;
if (gDvm.dexOptMode != OPTIMIZE_MODE_NONE) {
    flags |= DEXOPT_OPT_ENABLED;
    if (gDvm.dexOptMode == OPTIMIZE_MODE_ALL)
        flags |= DEXOPT_OPT_ALL;
}
if (gDvm.classVerifyMode != VERIFY_MODE_NONE) {
    flags |= DEXOPT_VERIFY_ENABLED;
    if (gDvm.classVerifyMode == VERIFY_MODE_ALL)
        flags |= DEXOPT_VERIFY_ALL;
}
if (isBootstrap)
    flags |= DEXOPT_IS_BOOTSTRAP;
if (gDvm.generateRegisterMaps)
    flags |= DEXOPT_GEN_REGISTER_MAPS;
sprintf(values[9], "%d", flags);
argv[curArg++] = values[9];

assert(((!kUseValgrind && curArg == kFixedArgCount) ||
        ((kUseValgrind && curArg == kFixedArgCount+kValgrindArgCount))));

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

ClassPathEntry* cpe;
for (cpe = gDvm.bootClassPath; cpe->ptr != NULL; cpe++) {
    argv[curArg++] = cpe->fileName;
}
assert(curArg == argc);

argv[curArg] = NULL;

if (kUseValgrind)
    execv(kValgrinder, const_cast<char**>(argv));
else
    execv(execFile, const_cast<char**>(argv));

ALOGE("execv '%s'%s failed: %s", execFile,
    kUseValgrind ? " [valgrind]" : "", strerror(errno));
exit(1);
} else {
    ALOGV("DexOpt: waiting for verify+opt, pid=%d", (int) pid);
    int status;
    pid_t gotPid;

    /*
     * Wait for the optimization process to finish. We go into VMWAIT
     * mode here so GC suspension won't have to wait for us.
     */
    ThreadStatus oldStatus = dvmChangeStatus(NULL, THREAD_VMWAIT);
    while (true) {
        gotPid = waitpid(pid, &status, 0);
        if (gotPid == -1 && errno == EINTR) {
            ALOGD("waitpid interrupted, retrying");
        } else {
            break;
        }
    }
    dvmChangeStatus(NULL, oldStatus);
    if (gotPid != pid) {
        ALOGE("waitpid failed: wanted %d, got %d: %s",
            (int) pid, (int) gotPid, strerror(errno));
        return false;
    }

    if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
        ALOGD("DexOpt: --- END '%s' (success) ---", lastPart);
        return true;
    } else {
        ALOGW("DexOpt: --- END '%s' --- status=0x%04x, process failed",
            lastPart, status);
        return false;
    }
}
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

}

/*
 * Do the actual optimization. This is executed in the dexopt process.
 *
 * For best use of disk/memory, we want to extract once and perform
 * optimizations in place. If the file has to expand or contract
 * to match local structure padding/alignment expectations, we want
 * to do the rewrite as part of the extract, rather than extracting
 * into a temp file and slurping it back out. (The structure alignment
 * is currently correct for all platforms, and this isn't expected to
 * change, so we should be okay with having it already extracted.)
 *
 * Returns "true" on success.
 */
bool dvmContinueOptimization(int fd, off_t dexOffset, long dexLength,
    const char* fileName, u4 modWhen, u4 crc, bool isBootstrap)
{
    DexClassLookup* pClassLookup = NULL;
    RegisterMapBuilder* pRegMapBuilder = NULL;

    assert(gDvm.optimizing);

    ALOGV("Continuing optimization (%s, isb=%d)", fileName, isBootstrap);

    assert(dexOffset >= 0);

    /* quick test so we don't blow up on empty file */
    if (dexLength < (int) sizeof(DexHeader)) {
        ALOGE("too small to be DEX");
        return false;
    }
    if (dexOffset < (int) sizeof(DexOptHeader)) {
        ALOGE("not enough room for opt header");
        return false;
    }
}

bool result = false;

/*
 * Drop this into a global so we don't have to pass it around. We could
 * also add a field to DexFile, but since it only pertains to DEX
 * creation that probably doesn't make sense.
 */
gDvm.optimizingBootstrapClass = isBootstrap;

{
    /*
     * Map the entire file (so we don't have to worry about page
     * alignment). The expectation is that the output file contains

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * our DEX data plus room for a small header.
    */
    bool success;
    void* mapAddr;
    mapAddr = mmap(NULL, dexOffset + dexLength, PROT_READ|PROT_WRITE,
        MAP_SHARED, fd, 0);
    if (mapAddr == MAP_FAILED) {
        ALOGE("unable to mmap DEX cache: %s", strerror(errno));
        goto bail;
    }

    bool doVerify, doOpt;
    if (gDvm.classVerifyMode == VERIFY_MODE_NONE) {
        doVerify = false;
    } else if (gDvm.classVerifyMode == VERIFY_MODE_REMOTE) {
        doVerify = !gDvm.optimizingBootstrapClass;
    } else /*if (gDvm.classVerifyMode == VERIFY_MODE_ALL)*/ {
        doVerify = true;
    }

    if (gDvm.dexOptMode == OPTIMIZE_MODE_NONE) {
        doOpt = false;
    } else if (gDvm.dexOptMode == OPTIMIZE_MODE_VERIFIED ||
        gDvm.dexOptMode == OPTIMIZE_MODE_FULL) {
        doOpt = doVerify;
    } else /*if (gDvm.dexOptMode == OPTIMIZE_MODE_ALL)*/ {
        doOpt = true;
    }

    /*
    * Rewrite the file. Byte reordering, structure realigning,
    * class verification, and bytecode optimization are all performed
    * here.
    *
    * In theory the file could change size and bits could shift around.
    * In practice this would be annoying to deal with, so the file
    * layout is designed so that it can always be rewritten in place.
    *
    * This creates the class lookup table as part of doing the processing.
    */
    success = rewriteDex(((u1*) mapAddr) + dexOffset, dexLength,
        doVerify, doOpt, &pClassLookup, NULL);

    if (success) {
        DvmDex* pDvmDex = NULL;
        u1* dexAddr = ((u1*) mapAddr) + dexOffset;

        if (dvmDexFileOpenPartial(dexAddr, dexLength, &pDvmDex) != 0) {
            ALOGE("Unable to create DexFile");
            success = false;
        }
    }

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    } else {
        /*
         * If configured to do so, generate register map output
         * for all verified classes. The register maps were
         * generated during verification, and will now be serialized.
         */
        if (gDvm.generateRegisterMaps) {
            pRegMapBuilder = dvmGenerateRegisterMaps(pDvmDex);
            if (pRegMapBuilder == NULL) {
                ALOGE("Failed generating register maps");
                success = false;
            }
        }

        DexHeader* pHeader = (DexHeader*)pDvmDex->pHeader;
        updateChecksum(dexAddr, dexLength, pHeader);

        dvmDexFileFree(pDvmDex);
    }
}

/* unmap the read-write version, forcing writes to disk */
if (msync(mapAddr, dexOffset + dexLength, MS_SYNC) != 0) {
    ALOGW("msync failed: %s", strerror(errno));
    // weird, but keep going
}
#ifdef 1
/*
 * This causes clean shutdown to fail, because we have loaded classes
 * that point into it. For the optimizer this isn't a problem,
 * because it's more efficient for the process to simply exit.
 * Exclude this code when doing clean shutdown for valgrind.
 */
if (munmap(mapAddr, dexOffset + dexLength) != 0) {
    ALOGE("munmap failed: %s", strerror(errno));
    goto bail;
}
#endif

if (!success)
    goto bail;
}

/* get start offset, and adjust deps start for 64-bit alignment */
off_t depsOffset, optOffset, endOffset, adjOffset;
int depsLength, optLength;
u4 optChecksum;

depsOffset = lseek(fd, 0, SEEK_END);
if (depsOffset < 0) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        ALOGE("lseek to EOF failed: %s", strerror(errno));
        goto bail;
    }
    adjOffset = (depsOffset + 7) & ~(0x07);
    if (adjOffset != depsOffset) {
        ALOGV("Adjusting deps start from %d to %d",
            (int) depsOffset, (int) adjOffset);
        depsOffset = adjOffset;
        lseek(fd, depsOffset, SEEK_SET);
    }

    /*
     * Append the dependency list.
     */
    if (writeDependencies(fd, modWhen, crc) != 0) {
        ALOGW("Failed writing dependencies");
        goto bail;
    }

    /* compute deps length, then adjust opt start for 64-bit alignment */
    optOffset = lseek(fd, 0, SEEK_END);
    depsLength = optOffset - depsOffset;

    adjOffset = (optOffset + 7) & ~(0x07);
    if (adjOffset != optOffset) {
        ALOGV("Adjusting opt start from %d to %d",
            (int) optOffset, (int) adjOffset);
        optOffset = adjOffset;
        lseek(fd, optOffset, SEEK_SET);
    }

    /*
     * Append any optimized pre-computed data structures.
     */
    if (!writeOptData(fd, pClassLookup, pRegMapBuilder)) {
        ALOGW("Failed writing opt data");
        goto bail;
    }

    endOffset = lseek(fd, 0, SEEK_END);
    optLength = endOffset - optOffset;

    /* compute checksum from start of deps to end of opt area */
    if (!computeFileChecksum(fd, depsOffset,
        (optOffset+optLength) - depsOffset, &optChecksum))
    {
        goto bail;
    }

    /*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * Output the "opt" header with all values filled in and a correct
    * magic number.
    */
    DexOptHeader optHdr;
    memset(&optHdr, 0xff, sizeof(optHdr));
    memcpy(optHdr.magic, DEX_OPT_MAGIC, 4);
    memcpy(optHdr.magic+4, DEX_OPT_MAGIC_VERS, 4);
    optHdr.dexOffset = (u4) dexOffset;
    optHdr.dexLength = (u4) dexLength;
    optHdr.depsOffset = (u4) depsOffset;
    optHdr.depsLength = (u4) depsLength;
    optHdr.optOffset = (u4) optOffset;
    optHdr.optLength = (u4) optLength;
    #if __BYTE_ORDER != __LITTLE_ENDIAN
        optHdr.flags = DEX_OPT_FLAG_BIG;
    #else
        optHdr.flags = 0;
    #endif
    optHdr.checksum = optChecksum;

    fsync(fd);    /* ensure previous writes go before header is written */

    lseek(fd, 0, SEEK_SET);
    if (sysWriteFully(fd, &optHdr, sizeof(optHdr), "DexOpt opt header") != 0)
        goto bail;

    ALOGV("Successfully wrote DEX header");
    result = true;

    //dvmRegisterMapDumpStats();

bail:
    dvmFreeRegisterMapBuilder(pRegMapBuilder);
    free(pClassLookup);
    return result;
}

/*
 * Prepare an in-memory DEX file.
 *
 * The data was presented to the VM as a byte array rather than a file.
 * We want to do the same basic set of operations, but we can just leave
 * them in memory instead of writing them out to a cached optimized DEX file.
 */
bool dvmPrepareDexInMemory(u1* addr, size_t len, DvmDex** ppDvmDex)
{
    DexClassLookup* pClassLookup = NULL;

    /*
     * Byte-swap, realign, verify basic DEX file structure.

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

*
* We could load + verify + optimize here as well, but that's probably
* not desirable.
*
* (The bulk-verification code is currently only setting the DEX
* file's "verified" flag, not updating the ClassObject. This would
* also need to be changed, or we will try to verify the class twice,
* and possibly reject it when optimized opcodes are encountered.)
*/
if (!rewriteDex(addr, len, false, false, &pClassLookup, ppDvmDex)) {
    return false;
}

(*ppDvmDex)->pDexFile->pClassLookup = pClassLookup;

return true;
}

/*
* Perform in-place rewrites on a memory-mapped DEX file.
*
* If this is called from a short-lived child process (dexopt), we can
* go nutty with loading classes and allocating memory. When it's
* called to prepare classes provided in a byte array, we may want to
* be more conservative.
*
* If "ppClassLookup" is non-NULL, a pointer to a newly-allocated
* DexClassLookup will be returned on success.
*
* If "ppDvmDex" is non-NULL, a newly-allocated DvmDex struct will be
* returned on success.
*/
static bool rewriteDex(u1* addr, int len, bool doVerify, bool doOpt,
    DexClassLookup** ppClassLookup, DvmDex** ppDvmDex)
{
    DexClassLookup* pClassLookup = NULL;
    u8 prepWhen, loadWhen, verifyOptWhen;
    DvmDex* pDvmDex = NULL;
    bool result = false;
    const char* msgStr = "???";

    /* if the DEX is in the wrong byte order, swap it now */
    if (dexSwapAndVerify(addr, len) != 0)
        goto bail;

    /*
    * Now that the DEX file can be read directly, create a DexFile struct
    * for it.
    */
    if (dvmDexFileOpenPartial(addr, len, &pDvmDex) != 0) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    ALOGE("Unable to create DexFile");
    goto bail;
}

/*
 * Create the class lookup table. This will eventually be appended
 * to the end of the .odex.
 *
 * We create a temporary link from the DexFile for the benefit of
 * class loading, below.
 */
pClassLookup = dexCreateClassLookup(pDvmDex->pDexFile);
if (pClassLookup == NULL)
    goto bail;
pDvmDex->pDexFile->pClassLookup = pClassLookup;

/*
 * If we're not going to attempt to verify or optimize the classes,
 * there's no value in loading them, so bail out early.
 */
if (!doVerify && !doOpt) {
    result = true;
    goto bail;
}

prepWhen = dvmGetRelativeTimeUsec();

/*
 * Load all classes found in this DEX file. If they fail to load for
 * some reason, they won't get verified (which is as it should be).
 */
if (!loadAllClasses(pDvmDex))
    goto bail;
loadWhen = dvmGetRelativeTimeUsec();

/*
 * Create a data structure for use by the bytecode optimizer.
 * We need to look up methods in a few classes, so this may cause
 * a bit of class loading. We usually do this during VM init, but
 * for dexopt on core.jar the order of operations gets a bit tricky,
 * so we defer it to here.
 */
if (!dvmCreateInlineSubsTable())
    goto bail;

/*
 * Verify and optimize all classes in the DEX file (command-line
 * options permitting).
 *
 * This is best-effort, so there's really no way for dexopt to

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * fail at this point.
    */
    verifyAndOptimizeClasses(pDvmDex->pDexFile, doVerify, doOpt);
    verifyOptWhen = dvmGetRelativeTimeUsec();

    if (doVerify && doOpt)
        msgStr = "verify+opt";
    else if (doVerify)
        msgStr = "verify";
    else if (doOpt)
        msgStr = "opt";
    ALOGD("DexOpt: load %dms, %s %dms, %d bytes",
        (int) (loadWhen - prepWhen) / 1000,
        msgStr,
        (int) (verifyOptWhen - loadWhen) / 1000,
        gDvm.pBootLoaderAlloc->curOffset);

    result = true;

bail:
    /*
     * On success, return the pieces that the caller asked for.
     */

    if (pDvmDex != NULL) {
        /* break link between the two */
        pDvmDex->pDexFile->pClassLookup = NULL;
    }

    if (ppDvmDex == NULL || !result) {
        dvmDexFileFree(pDvmDex);
    } else {
        *ppDvmDex = pDvmDex;
    }

    if (ppClassLookup == NULL || !result) {
        free(pClassLookup);
    } else {
        *ppClassLookup = pClassLookup;
    }

    return result;
}

/*
 * Try to load all classes in the specified DEX. If they have some sort
 * of broken dependency, e.g. their superclass lives in a different DEX
 * that wasn't previously loaded into the bootstrap class path, loading
 * will fail. This is the desired behavior.
 */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* We have no notion of class loader at this point, so we load all of
* the classes with the bootstrap class loader. It turns out this has
* exactly the behavior we want, and has no ill side effects because we're
* running in a separate process and anything we load here will be forgotten.
*
* We set the CLASS_MULTIPLE_DEFS flag here if we see multiple definitions.
* This works because we only call here as part of optimization / pre-verify,
* not during verification as part of loading a class into a running VM.
*
* This returns "false" if the world is too screwed up to do anything
* useful at all.
*/
static bool loadAllClasses(DvmDex* pDvmDex)
{
    u4 count = pDvmDex->pDexFile->pHeader->classDefsSize;
    u4 idx;
    int loaded = 0;

    ALOGV("DexOpt: +++ trying to load %d classes", count);

    dvmSetBootPathExtraDex(pDvmDex);

    /*
    * At this point, it is safe -- and necessary! -- to look up the
    * VM's required classes and members, even when what we are in the
    * process of processing is the core library that defines these
    * classes itself. (The reason it is necessary is that in the act
    * of initializing the class Class, below, the system will end up
    * referring to many of the class references that got set up by
    * this call.)
    */
    if (!dvmFindRequiredClassesAndMembers()) {
        return false;
    }

    /*
    * We have some circularity issues with Class and Object that are
    * most easily avoided by ensuring that Object is never the first
    * thing we try to find-and-initialize. The call to
    * dvmFindSystemClass() here takes care of that situation. (We
    * only need to do this when loading classes from the DEX file
    * that contains Object, and only when Object comes first in the
    * list, but it costs very little to do it in all cases.)
    */
    if (!dvmInitClass(gDvm.classJavaLangClass)) {
        ALOGE("ERROR: failed to initialize the class Class!");
        return false;
    }

    for (idx = 0; idx < count; idx++) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

const DexClassDef* pClassDef;
const char* classDescriptor;
ClassObject* newClass;

pClassDef = dexGetClassDef(pDvmDex->pDexFile, idx);
classDescriptor =
    dexStringByTypeIdx(pDvmDex->pDexFile, pClassDef->classIdx);

ALOGV("+++ loading '%s'", classDescriptor);
//newClass = dvmDefineClass(pDexFile, classDescriptor,
//    NULL);
newClass = dvmFindSystemClassNoInit(classDescriptor);
if (newClass == NULL) {
    ALOGV("DexOpt: failed loading '%s'", classDescriptor);
    dvmClearOptException(dvmThreadSelf());
} else if (newClass->pDvmDex != pDvmDex) {
    /*
     * We don't load the new one, and we tag the first one found
     * with the "multiple def" flag so the resolver doesn't try
     * to make it available.
     */
    ALOGD("DexOpt: '%s' has an earlier definition; blocking out",
        classDescriptor);
    SET_CLASS_FLAG(newClass, CLASS_MULTIPLE_DEFS);
} else {
    loaded++;
}
}
ALOGV("DexOpt: +++ successfully loaded %d classes", loaded);

dvmSetBootPathExtraDex(NULL);
return true;
}

/*
 * Verify and/or optimize all classes that were successfully loaded from
 * this DEX file.
 */
static void verifyAndOptimizeClasses(DexFile* pDexFile, bool doVerify,
    bool doOpt)
{
    u4 count = pDexFile->pHeader->classDefsSize;
    u4 idx;

    for (idx = 0; idx < count; idx++) {
        const DexClassDef* pClassDef;
        const char* classDescriptor;
        ClassObject* clazz;

        pClassDef = dexGetClassDef(pDexFile, idx);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

classDescriptor = dexStringByTypeIdx(pDexFile, pClassDef->classIdx);

/* all classes are loaded into the bootstrap class loader */
clazz = dvmLookupClass(classDescriptor, NULL, false);
if (clazz != NULL) {
    verifyAndOptimizeClass(pDexFile, clazz, pClassDef, doVerify, doOpt);
} else {
    // TODO: log when in verbose mode
    ALOGV("DexOpt: not optimizing unavailable class '%s'",
        classDescriptor);
}
}

#ifdef VERIFIER_STATS
ALOGI("Verifier stats:");
ALOGI(" methods examined      : %u", gDvm.verifierStats.methodsExamined);
ALOGI(" monitor-enter methods   : %u", gDvm.verifierStats.monEnterMethods);
ALOGI(" instructions examined    : %u", gDvm.verifierStats.instrsExamined);
ALOGI(" instructions re-examined: %u", gDvm.verifierStats.instrsReexamined);
ALOGI(" copying of register sets: %u", gDvm.verifierStats.copyRegCount);
ALOGI(" merging of register sets: %u", gDvm.verifierStats.mergeRegCount);
ALOGI(" ...that caused changes   : %u", gDvm.verifierStats.mergeRegChanged);
ALOGI(" uninit searches         : %u", gDvm.verifierStats.uninitSearches);
ALOGI(" max memory required     : %u", gDvm.verifierStats.biggestAlloc);
#endif
}

/*
 * Verify and/or optimize a specific class.
 */
static void verifyAndOptimizeClass(DexFile* pDexFile, ClassObject* clazz,
    const DexClassDef* pClassDef, bool doVerify, bool doOpt)
{
    const char* classDescriptor;
    bool verified = false;

    if (clazz->pDvmDex->pDexFile != pDexFile) {
        /*
         * The current DEX file defined a class that is also present in the
         * bootstrap class path. The class loader favored the bootstrap
         * version, which means that we have a pointer to a class that is
         * (a) not the one we want to examine, and (b) mapped read-only,
         * so we will seg fault if we try to rewrite instructions inside it.
         */
        ALOGD("DexOpt: not verifying/optimizing '%s': multiple definitions",
            clazz->descriptor);
        return;
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

classDescriptor = dexStringByTypeIdx(pDexFile, pClassDef->classIdx);

/*
 * First, try to verify it.
 */
if (doVerify) {
    if (dvmVerifyClass(clazz)) {
        /*
         * Set the "is preverified" flag in the DexClassDef. We
         * do it here, rather than in the ClassObject structure,
         * because the DexClassDef is part of the odex file.
         */
        assert((clazz->accessFlags & JAVA_FLAGS_MASK) ==
            pClassDef->accessFlags);
        ((DexClassDef*)pClassDef)->accessFlags |= CLASS_ISPREVERIFIED;
        verified = true;
    } else {
        // TODO: log when in verbose mode
        ALOGV("DexOpt: '%s' failed verification", classDescriptor);
    }
}

if (doOpt) {
    bool needVerify = (gDvm.dexOptMode == OPTIMIZE_MODE_VERIFIED ||
        gDvm.dexOptMode == OPTIMIZE_MODE_FULL);
    if (!verified && needVerify) {
        ALOGV("DexOpt: not optimizing '%s': not verified",
            classDescriptor);
    } else {
        dvmOptimizeClass(clazz, false);

        /* set the flag whether or not we actually changed anything */
        ((DexClassDef*)pClassDef)->accessFlags |= CLASS_ISOPTIMIZED;
    }
}
}

/*
 * Get the cache file name from a ClassPathEntry.
 */
static const char* getCacheFileName(const ClassPathEntry* cpe)
{
    switch (cpe->kind) {
    case kCpeJar:
        return dvmGetJarFileCacheFileName((JarFile*) cpe->ptr);
    case kCpeDex:
        return dvmGetRawDexFileCacheFileName((RawDexFile*) cpe->ptr);
    default:
        ALOGE("DexOpt: unexpected cpe kind %d", cpe->kind);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        dvmAbort();
        return NULL;
    }
}

/*
 * Get the SHA-1 signature.
 */
static const u1* getSignature(const ClassPathEntry* cpe)
{
    DvmDex* pDvmDex;

    switch (cpe->kind) {
    case kCpeJar:
        pDvmDex = dvmGetJarFileDex((JarFile*) cpe->ptr);
        break;
    case kCpeDex:
        pDvmDex = dvmGetRawDexFileDex((RawDexFile*) cpe->ptr);
        break;
    default:
        ALOGE("unexpected cpe kind %d", cpe->kind);
        dvmAbort();
        pDvmDex = NULL;    // make gcc happy
    }

    assert(pDvmDex != NULL);
    return pDvmDex->pDexFile->pHeader->signature;
}

/*
 * Dependency layout:
 * 4b Source file modification time, in seconds since 1970 UTC
 * 4b CRC-32 from Zip entry, or Adler32 from source DEX header
 * 4b Dalvik VM build number
 * 4b Number of dependency entries that follow
 * Dependency entries:
 * 4b Name length (including terminating null)
 * var Full path of cache entry (null terminated)
 * 20b SHA-1 signature from source DEX file
 *
 * If this changes, update DEX_OPT_MAGIC_VERS.
 */
static const size_t kMinDepSize = 4 * 4;
static const size_t kMaxDepSize = 4 * 4 + 2048;    // sanity check

/*
 * Read the "opt" header, verify it, then read the dependencies section
 * and verify that data as well.
 */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* If "sourceAvail" is "true", this will verify that "modWhen" and "crc"
* match up with what is stored in the header. If they don't, we reject
* the file so that it can be recreated from the updated original. If
* "sourceAvail" isn't set, e.g. for a .odex file, we ignore these arguments.
*
* On successful return, the file will be seeked immediately past the
* "opt" header.
*/
bool dvmCheckOptHeaderAndDependencies(int fd, bool sourceAvail, u4 modWhen,
u4 crc, bool expectVerify, bool expectOpt)
{
    DexOptHeader optHdr;
    u1* depData = NULL;
    const u1* magic;
    off_t posn;
    int result = false;
    ssize_t actual;

    /*
    * Start at the start. The "opt" header, when present, will always be
    * the first thing in the file.
    */
    if (lseek(fd, 0, SEEK_SET) != 0) {
        ALOGE("DexOpt: failed to seek to start of file: %s", strerror(errno));
        goto bail;
    }

    /*
    * Read and do trivial verification on the opt header. The header is
    * always in host byte order.
    */
    actual = read(fd, &optHdr, sizeof(optHdr));
    if (actual < 0) {
        ALOGE("DexOpt: failed reading opt header: %s", strerror(errno));
        goto bail;
    } else if (actual != sizeof(optHdr)) {
        ALOGE("DexOpt: failed reading opt header (got %d of %zd)",
            (int) actual, sizeof(optHdr));
        goto bail;
    }

    magic = optHdr.magic;
    if (memcmp(magic, DEX_MAGIC, 4) == 0) {
        /* somebody probably pointed us at the wrong file */
        ALOGD("DexOpt: expected optimized DEX, found unoptimized");
        goto bail;
    } else if (memcmp(magic, DEX_OPT_MAGIC, 4) != 0) {
        /* not a DEX file, or previous attempt was interrupted */
        ALOGD("DexOpt: incorrect opt magic number (0x%02x %02x %02x %02x)",
            magic[0], magic[1], magic[2], magic[3]);
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    goto bail;
}
if (memcmp(magic+4, DEX_OPT_MAGIC_VERS, 4) != 0) {
    ALOGW("DexOpt: stale opt version (0x%02x %02x %02x %02x)",
        magic[4], magic[5], magic[6], magic[7]);
    goto bail;
}
if (optHdr.depsLength < kMinDepSize || optHdr.depsLength > kMaxDepSize) {
    ALOGW("DexOpt: weird deps length %d, bailing", optHdr.depsLength);
    goto bail;
}

/*
 * Do the header flags match up with what we want?
 *
 * The only thing we really can't handle is incorrect byte ordering.
 */
{
    const u4 matchMask = DEX_OPT_FLAG_BIG;
    u4 expectedFlags = 0;
#ifdef __BYTE_ORDER != __LITTLE_ENDIAN
    expectedFlags |= DEX_OPT_FLAG_BIG;
#endif
    if ((expectedFlags & matchMask) != (optHdr.flags & matchMask)) {
        ALOGI("DexOpt: header flag mismatch (0x%02x vs 0x%02x, mask=0x%02x)",
            expectedFlags, optHdr.flags, matchMask);
        goto bail;
    }
}

posn = lseek(fd, optHdr.depsOffset, SEEK_SET);
if (posn < 0) {
    ALOGW("DexOpt: seek to deps failed: %s", strerror(errno));
    goto bail;
}

/*
 * Read all of the dependency stuff into memory.
 */
depData = (u1*) malloc(optHdr.depsLength);
if (depData == NULL) {
    ALOGW("DexOpt: unable to allocate %d bytes for deps",
        optHdr.depsLength);
    goto bail;
}
actual = read(fd, depData, optHdr.depsLength);
if (actual < 0) {
    ALOGW("DexOpt: failed reading deps: %s", strerror(errno));
    goto bail;
} else if (actual != (ssize_t) optHdr.depsLength) {

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    ALOGW("DexOpt: failed reading deps: got %d of %d",
          (int) actual, optHdr.depsLength);
    goto bail;
}

/*
 * Verify simple items.
 */
const u1* ptr;
u4 val;

ptr = depData;
val = read4LE(&ptr);
if (sourceAvail && val != modWhen) {
    ALOGI("DexOpt: source file mod time mismatch (%08x vs %08x)",
          val, modWhen);
    goto bail;
}
val = read4LE(&ptr);
if (sourceAvail && val != crc) {
    ALOGI("DexOpt: source file CRC mismatch (%08x vs %08x)", val, crc);
    goto bail;
}
val = read4LE(&ptr);
if (val != DALVIK_VM_BUILD) {
    ALOGD("DexOpt: VM build version mismatch (%d vs %d)",
          val, DALVIK_VM_BUILD);
    goto bail;
}

/*
 * Verify dependencies on other cached DEX files. It must match
 * exactly with what is currently defined in the bootclasspath.
 */
ClassPathEntry* cpe;
u4 numDeps;

numDeps = read4LE(&ptr);
ALOGV("+++ DexOpt: numDeps = %d", numDeps);
for (cpe = gDvm.bootClassPath; cpe->ptr != NULL; cpe++) {
    const char* cacheFileName =
        dvmPathToAbsolutePortion(getCacheFileName(cpe));
    assert(cacheFileName != NULL); /* guaranteed by Class.c */

    const u1* signature = getSignature(cpe);
    size_t len = strlen(cacheFileName) + 1;
    u4 storedStrLen;

    if (numDeps == 0) {
        /* more entries in bootclasspath than in deps list */

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        ALOGI("DexOpt: not all deps represented");
        goto bail;
    }

    storedStrLen = read4LE(&ptr);
    if (len != storedStrLen ||
        strcmp(cacheFileName, (const char*) ptr) != 0)
    {
        ALOGI("DexOpt: mismatch dep name: '%s' vs. '%s'",
            cacheFileName, ptr);
        goto bail;
    }

    ptr += storedStrLen;

    if (memcmp(signature, ptr, kSHA1DigestLen) != 0) {
        ALOGI("DexOpt: mismatch dep signature for '%s'", cacheFileName);
        goto bail;
    }
    ptr += kSHA1DigestLen;

    ALOGV("DexOpt: dep match on '%s'", cacheFileName);

    numDeps--;
}

if (numDeps != 0) {
    /* more entries in deps list than in classpath */
    ALOGI("DexOpt: Some deps went away");
    goto bail;
}

// consumed all data and no more?
if (ptr != depData + optHdr.depsLength) {
    ALOGW("DexOpt: Spurious dep data? %d vs %d",
        (int) (ptr - depData), optHdr.depsLength);
    assert(false);
}

result = true;

bail:
    free(depData);
    return result;
}

/*
 * Write the dependency info to "fd" at the current file position.
 */
static int writeDependencies(int fd, u4 modWhen, u4 crc)

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

{
    u1* buf = NULL;
    int result = -1;
    ssize_t bufLen;
    ClassPathEntry* cpe;
    int numDeps;

    /*
     * Count up the number of completed entries in the bootclasspath.
     */
    numDeps = 0;
    bufLen = 0;
    for (cpe = gDvm.bootClassPath; cpe->ptr != NULL; cpe++) {
        const char* cacheFileName =
            dvmPathToAbsolutePortion(getCacheFileName(cpe));
        assert(cacheFileName != NULL); /* guaranteed by Class.c */

        ALOGV("+++ DexOpt: found dep '%s'", cacheFileName);

        numDeps++;
        bufLen += strlen(cacheFileName) + 1;
    }

    bufLen += 4*4 + numDeps * (4+kSHA1DigestLen);

    buf = (u1*)malloc(bufLen);

    set4LE(buf+0, modWhen);
    set4LE(buf+4, crc);
    set4LE(buf+8, DALVIK_VM_BUILD);
    set4LE(buf+12, numDeps);

    // TODO: do we want to add dvmGetInlineOpsTableLength() here? Won't
    // help us if somebody replaces an existing entry, but it'd catch
    // additions/removals.

    u1* ptr = buf + 4*4;
    for (cpe = gDvm.bootClassPath; cpe->ptr != NULL; cpe++) {
        const char* cacheFileName =
            dvmPathToAbsolutePortion(getCacheFileName(cpe));
        assert(cacheFileName != NULL); /* guaranteed by Class.c */

        const u1* signature = getSignature(cpe);
        int len = strlen(cacheFileName) + 1;

        if (ptr + 4 + len + kSHA1DigestLen > buf + bufLen) {
            ALOGE("DexOpt: overran buffer");
            dvmAbort();
        }
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

        set4LE(ptr, len);
        ptr += 4;
        memcpy(ptr, cacheFileName, len);
        ptr += len;
        memcpy(ptr, signature, kSHA1DigestLen);
        ptr += kSHA1DigestLen;
    }

    assert(ptr == buf + bufLen);

    result = sysWriteFully(fd, buf, bufLen, "DexOpt dep info");

    free(buf);
    return result;
}

/*
 * Write a block of data in "chunk" format.
 *
 * The chunk header fields are always in "native" byte order.  If "size"
 * is not a multiple of 8 bytes, the data area is padded out.
 */
static bool writeChunk(int fd, u4 type, const void* data, size_t size)
{
    union {                /* save a syscall by grouping these together */
        char raw[8];
        struct {
            u4 type;
            u4 size;
        } ts;
    } header;

    assert(sizeof(header) == 8);

    ALOGV("Writing chunk, type=%.4s size=%d", (char*) &type, size);

    header.ts.type = type;
    header.ts.size = (u4) size;
    if (sysWriteFully(fd, &header, sizeof(header),
        "DexOpt opt chunk header write") != 0)
    {
        return false;
    }

    if (size > 0) {
        if (sysWriteFully(fd, data, size, "DexOpt opt chunk write") != 0)
            return false;
    }
}

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

/* if necessary, pad to 64-bit alignment */
if ((size & 7) != 0) {
    int padSize = 8 - (size & 7);
    ALOGV("size was %d, inserting %d pad bytes", size, padSize);
    lseek(fd, padSize, SEEK_CUR);
}

assert( ((int)lseek(fd, 0, SEEK_CUR) & 7) == 0);

return true;
}

/*
 * Write opt data.
 *
 * We have different pieces, some of which may be optional. To make the
 * most effective use of space, we use a "chunk" format, with a 4-byte
 * type and a 4-byte length. We guarantee 64-bit alignment for the data,
 * so it can be used directly when the file is mapped for reading.
 */
static bool writeOptData(int fd, const DexClassLookup* pClassLookup,
    const RegisterMapBuilder* pRegMapBuilder)
{
    /* pre-computed class lookup hash table */
    if (!writeChunk(fd, (u4) kDexChunkClassLookup,
        pClassLookup, pClassLookup->size))
    {
        return false;
    }

    /* register maps (optional) */
    if (pRegMapBuilder != NULL) {
        if (!writeChunk(fd, (u4) kDexChunkRegisterMaps,
            pRegMapBuilder->data, pRegMapBuilder->size))
        {
            return false;
        }
    }

    /* write the end marker */
    if (!writeChunk(fd, (u4) kDexChunkEnd, NULL, 0)) {
        return false;
    }

    return true;
}

/*
 * Compute a checksum on a piece of an open file.
 *
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* File will be positioned at end of checksummed area.
*
* Returns "true" on success.
*/
static bool computeFileChecksum(int fd, off_t start, size_t length, u4* pSum)
{
    unsigned char readBuf[8192];
    ssize_t actual;
    uLong Adler;

    if (lseek(fd, start, SEEK_SET) != start) {
        ALOGE("Unable to seek to start of checksum area (%ld): %s",
            (long) start, strerror(errno));
        return false;
    }

    Adler = Adler32(0L, Z_NULL, 0);

    while (length != 0) {
        size_t wanted = (length < sizeof(readBuf)) ? length : sizeof(readBuf);
        actual = read(fd, readBuf, wanted);
        if (actual <= 0) {
            ALOGE("Read failed (%d) while computing checksum (len=%zu): %s",
                (int) actual, length, strerror(errno));
            return false;
        }

        Adler = Adler32(Adler, readBuf, actual);

        length -= actual;
    }

    *pSum = Adler;
    return true;
}

/*
* Update the Adler-32 checksum stored in the DEX file. This covers the
* swapped and optimized DEX data, but does not include the opt header
* or optimized data.
*/
static void updateChecksum(u1* addr, int len, DexHeader* pHeader)
{
    /*
    * Rewrite the checksum. We leave the SHA-1 signature alone.
    */
    uLong Adler = Adler32(0L, Z_NULL, 0);
    const int nonSum = sizeof(pHeader->magic) + sizeof(pHeader->checksum);

    Adler = Adler32(Adler, addr + nonSum, len - nonSum);

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```
pHeader->checksum = adler;
}
```

```
xxxviii platform_dalvik-master\vm\oo\Object.h
```

```
/*
 * Class objects have many additional fields. This is used for both
 * classes and interfaces, including synthesized classes (arrays and
 * primitive types).
 *
 * Class objects are unusual in that they have some fields allocated with
 * the system malloc (or LinearAlloc), rather than on the GC heap. This is
 * handy during initialization, but does require special handling when
 * discarding java.lang.Class objects.
 *
 * The separation of methods (direct vs. virtual) and fields (class vs.
 * instance) used in Dalvik works out pretty well. The only time it's
 * annoying is when enumerating or searching for things with reflection.
 */
structClassObject : Object {
/* leave space for instance data; we could access fields directly if we
   freeze the definition of java/lang/Class */
u4instanceData[CLASS_FIELD_SLOTS];

/* UTF-8 descriptor for the class; from constant pool, or on heap
   if generated ("[C") */
constchar* descriptor;
char* descriptorAlloc;

/* access flags; low 16 bits are defined by VM spec */
u4accessFlags;

/* VM-unique class serial number, nonzero, set very early */
u4serialNumber;

/* DexFile from which we came; needed to resolve constant pool entries */
/* (will be NULL for VM-generated, e.g. arrays and primitive classes) */
DvmDex* pDvmDex;

/* state of class initialization */
ClassStatus status;

/* if class verify fails, we must return same error on subsequent tries */
ClassObject* verifyErrorClass;

/* threadId, used to check for recursive <clinit> invocation */
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

u4initThreadld;

/*
 * Total object size; used when allocating storage on gc heap. (For
 * interfaces and abstract classes this will be zero.)
 */
size_t      objectSize;

/* arrays only: class object for base element, for instanceof/checkcast
   (for String[][][], this will be String) */
ClassObject* elementClass;

/* arrays only: number of dimensions, e.g. int[][] is 2 */
intarrayDim;

/* primitive type index, or PRIM_NOT (-1); set for generated prim classes */
PrimitiveType primitiveType;

/* superclass, or NULL if this is java.lang.Object */
ClassObject* super;

/* defining class loader, or NULL for the "bootstrap" system loader */
Object*      classLoader;

/* initiating class loader list */
/* NOTE: for classes with low serialNumber, these are unused, and the
   values are kept in a table in gDvm. */
InitiatingLoaderListinitiatingLoaderList;

/* array of interfaces this class implements directly */
intinterfaceCount;
ClassObject** interfaces;

/* static, private, and <init> methods */
intdirectMethodCount;
Method*      directMethods;

/* virtual methods defined in this class; invoked through vtable */
intvirtualMethodCount;
Method*      virtualMethods;

/*
 * Virtual method table (vtable), for use by "invoke-virtual". The
 * vtable from the superclass is copied in, and virtual methods from
 * our class either replace those from the super or are appended.
 */
intvtableCount;
Method**      vtable;

/*

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

* Interface table (iftable), one entry per interface supported by
* this class. That means one entry for each interface we support
* directly, indirectly via superclass, or indirectly via
* superinterface. This will be null if neither we nor our superclass
* implement any interfaces.
*
* Why we need this: given "class Foo implements Face", declare
* "Face faceObj = new Foo()". Invoke faceObj.blah(), where "blah" is
* part of the Face interface. We can't easily use a single vtable.
*
* For every interface a concrete class implements, we create a list of
* virtualMethod indices for the methods in the interface.
*/
int iftableCount;
InterfaceEntry* iftable;

/*
* The interface vtable indices for iftable get stored here. By placing
* them all in a single pool for each class that implements interfaces,
* we decrease the number of allocations.
*/
int ifviPoolCount;
int* ifviPool;

/* instance fields
*
* These describe the layout of the contents of a DataObject-compatible
* Object. Note that only the fields directly defined by this class
* are listed in ifields; fields defined by a superclass are listed
* in the superclass's ClassObject.ifields.
*
* All instance fields that refer to objects are guaranteed to be
* at the beginning of the field list. ifieldRefCount specifies
* the number of reference fields.
*/
int ifieldCount;
int ifieldRefCount; // number of fields that are object refs
InstField* ifields;

/* bitmap of offsets of ifields */
u4refOffsets;

/* source file name, if known */
constchar* sourceFile;

/* static fields */
int sfieldCount;
StaticFieldsfields[0]; /* MUST be last item */
};

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

xxxix    platform_dalvik-master\vm\oo\Object.h

/*
 * A method. We create one of these for every method in every class
 * we load, so try to keep the size to a minimum.
 *
 * Much of this comes from and could be accessed in the data held in shared
 * memory. We hold it all together here for speed. Everything but the
 * pointers could be held in a shared table generated by the optimizer;
 * if we're willing to convert them to offsets and take the performance
 * hit (e.g. "meth->insns" becomes "baseAddr + meth->insnsOffset") we
 * could move everything but "nativeFunc".
 */
struct Method {
/* the class we are a part of */
ClassObject*  clazz;

/* access flags; low 16 bits are defined by spec (could be u2?) */
u4accessFlags;

/*
 * For concrete virtual methods, this is the offset of the method
 * in "vtable".
 *
 * For abstract methods in an interface class, this is the offset
 * of the method in "iftable[n]->methodIndexArray".
 */
u2methodIndex;

/*
 * Method bounds; not needed for an abstract method.
 *
 * For a native method, we compute the size of the argument list, and
 * set "insSize" and "registerSize" equal to it.
 */
u2registersSize; /* ins + locals */
u2outsSize;
u2insSize;

/* method name, e.g. "<init>" or "eatLunch" */
constchar*  name;

/*
 * Method prototype descriptor string (return and argument types).
 *
 * TODO: This currently must specify the DexFile as well as the proto_ids
 * index, because generated Proxy classes don't have a DexFile. We can
 * remove the DexFile* and reduce the size of this struct if we generate

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**


---

```

    * a DEX for proxies.
    */
DexProtoprototype;

/* short-form method descriptor string */
constchar*    shorty;

/*
    * The remaining items are not used for abstract or native methods.
    * (JNI is currently hijacking "insns" as a function pointer, set
    * after the first call. For internal-native this stays null.)
    */

/* the actual code */
constu2*    insns;    /* instructions, in memory-mapped .dex */

/* JNI: cached argument and return-type hints */
intjniArgInfo;

/*
    * JNI: native method ptr; could be actual function or a JNI bridge. We
    * don't currently discriminate between DalvikBridgeFunc and
    * DalvikNativeFunc; the former takes an argument superset (i.e. two
    * extra args) which will be ignored. If necessary we can use
    * insns==NULL to detect JNI bridge vs. internal native.
    */
DalvikBridgeFuncnativeFunc;

/*
    * JNI: true if this static non-synchronized native method (that has no
    * reference arguments) needs a JNIEnv* and jclass/object. Libcore
    * uses this.
    */
boolfastJni;

/*
    * JNI: true if this method has no reference arguments. This lets the JNI
    * bridge avoid scanning the shorty for direct pointers that need to be
    * converted to local references.
    *
    * TODO: replace this with a list of indexes of the reference arguments.
    */
boolnoRef;

/*
    * JNI: true if we should log entry and exit. This is the only way
    * developers can log the local references that are passed into their code.
    * Used for debugging JNI problems in third-party code.
    */
boolshouldTrace;

```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

---

```
/*  
 * Register map data, if available. This will point into the DEX file  
 * if the data was computed during pre-verification, or into the  
 * linear alloc area if not.  
 */  
constRegisterMap* registerMap;  
  
/* set if method was called during method profiling */  
boolinProfile;  
};
```

**CONFIDENTIAL – OUTSIDE ATTORNEYS EYES ONLY**

IN THE UNITED STATES DISTRICT COURT  
FOR THE WESTERN DISTRICT OF WASHINGTON

SEATTLE DIVISION

**Appendix B**

<b><i>See the documents identified below:</i></b>	<b>Corresponding mobile phones and/or tablets:</b>
REC_DEL_AeroATT_0000248-253	Aero AT&T
REC_DEL_Strk5ATT_0000005-11	Streak 5 AT&T
REC_DEL_Strk7NoC_0000005-9	Streak 7 No Carrier Version
REC_DEL_Strk7TMob_0000223-230	Streak 7 T-Mobile
REC_DEL_VenueNoC_0000184-190	Venue No Carrier
REC_DEL_AeroATT_0000248-253	Aero AT&T
REC_DEL_Strk5ATT_0000005-11	Streak 5 AT&T
REC_DEL_Strk7NoC_0000005-9	Streak 7 No Carrier Version
REC_DEL_Strk7TMob_0000223-230	Streak 7 T-Mobile